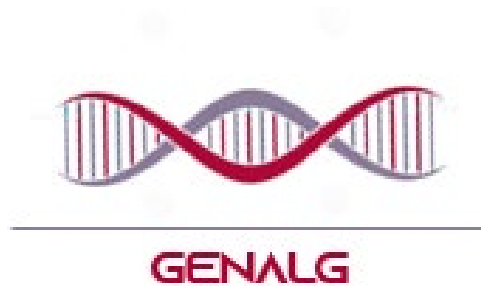


## Rapport Groupe 3 jeu évolutionnaire

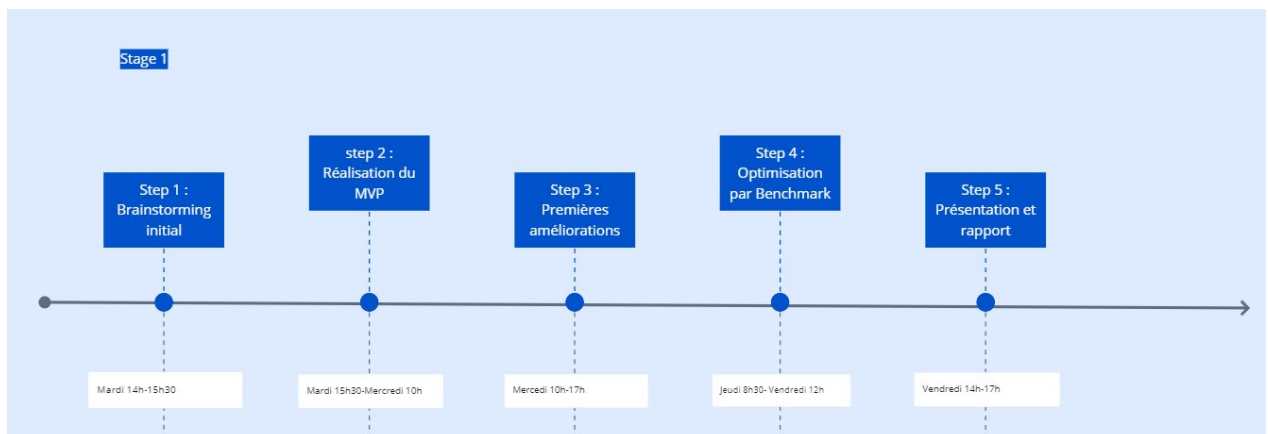
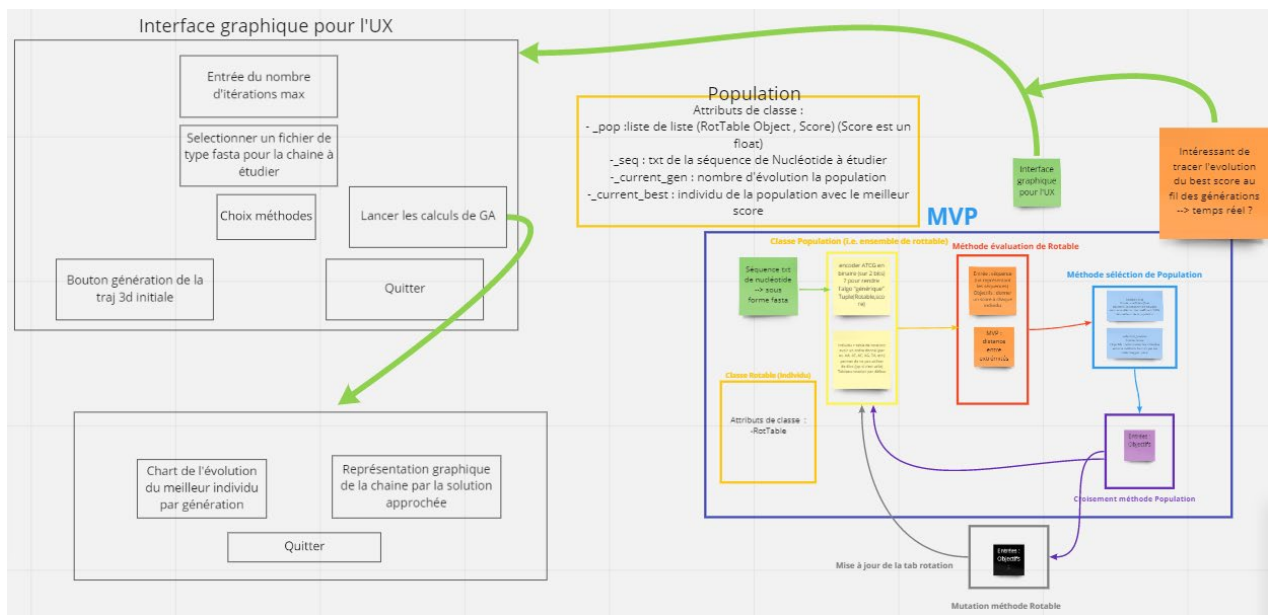


Sujet : Pour une séquence ADN donnée, jouer sur les angles entre les différents dinucléotides afin de tenter que l'ADN soit le plus proche possible de boucler sur lui-même et en gardant une complexité temporelle raisonnable.

Contraintes : les angles entre les différents dinucléotides doivent être réalistes chimiquement et sont bornés entre certaines valeurs, et il y a une certaine symétrie à conserver dans ces valeurs pour coller à la réalité biologique également.

### *I-Méthode d'analyse, organisation du travail et outils utilisés*

Le premier pas vers l'analyse du sujet fut de s'accorder sur le plan à suivre pour la construction de notre code. Grâce à l'outil miro nous avons pu organiser en étapes cette construction : [https://miro.com/app/board/o9J\\_lXQ3JWY=/](https://miro.com/app/board/o9J_lXQ3JWY=/). Pour ce qui est de l'organisation du travail, la majorité du travail a été effectué en groupe grâce à l'outil liveshare de Visual Studio Code. Ce mode de travail permettait une réflexion de groupe permanente sur tous les bouts de code écrit, il permettait également d'éviter les erreurs d'inattention dans les algorithmes et enfin à tout moment tous les membres du groupe avaient une image claire de ce que chaque fonction faisait et des changements qu'elle apportait.



Pour ce qui est de la découpe du travail en étape, le premier objectif était la réalisation d'un MVP : obtenir par l'utilisation d'un algorithme génétique sans mutation pour les valeurs d'angle mais avec des croisements entre les valeurs d'angle des dinucléotides des individus de la population ; la symétrie imposée par la forme réelle des brins d'ADN ne sera pas prise en compte dans ce MVP ni les bornes dans lesquelles sont comprises les angles. Puis petit à petit seront ajoutés les fonctionnalités permettant de respecter les contraintes du sujet, d'optimiser le code en temps et en résultat et de rendre plus esthétiques les résultats présentés.

## II-Structure et fonctionnement de l'algorithme

### 1) Le MVP : Un squelette d'algorithme génétique fonctionnel

Pour ce MVP l'algorithme était composé de 1 fichier créé par notre groupe qui est le fichier Population à l'intérieur duquel est défini la classe d'une population (d'une liste de tuple d'une table de rotation et d'un score) et 3 fichiers donnés, le fichier RotTable à l'intérieur duquel est défini la classe d'un individu (d'une table de rotation), le fichier main et le fichier traj3D. Ce MVP visait à avoir un squelette de l'algorithme final fonctionnel et utilisant des caractéristiques tirées de la théorie des algorithmes génétiques.

Deux méthodes sont ajoutées dans la classe RotTable pour ce MVP : Cross, qui permet de croiser les valeurs d'angle pour un dinucléotide pour 2 individus, et Evaluation, qui calcule la trajectoire de la séquence d'ADN pour une table de rotation et retourne le score associé à l'individu (correspond pour ce MVP à la distance définie à partir de la valeur absolue entre le premier et le dernier dinucléotide du brin d'ADN).

Pour la classe population, différentes méthodes ont d'abord été impliquées dans le but d'avoir des méthodes pour les opérations opérées plusieurs fois, exemple, mettre à jour la génération actuelle ou vider la population on initialise aussi notre population avec un nombre N d'individu et un score None pour chacun de ces individus. Ensuite, on crée des méthodes qui permettent la construction d'un algorithme génétique. Premièrement une méthode qui va chercher dans la classe RotTable le score de chaque table de rotation de notre objet et les rentrer dans les tuples. Deuxièmement, une méthode de sélection qui garde la moitié des individus d'une population (ceux avec le score le plus bas). Troisièmement, une méthode cross\_pop qui utilise la méthode cross de la table des individus et l'utilise sur les éléments de la population sélectionnée, on double donc la population et dans le cas d'une population paire on revient au nombre initial que l'on avait pour la taille de la population. Et enfin une méthode next\_gen qui lie les méthodes énoncées précédemment et permet à une la méthode eval de faire une boucle sur un nombre de génération de n et applique à chaque fois cet algorithme génétique sauf à la dernière génération où on évalue juste les scores des individus et on retourne le meilleur individu obtenu et son score.

Avec ce MVP on obtient déjà des premiers résultats justes dans le sens où on observe bien en affichant le meilleur score à chaque génération qu'il diminue et qu'on obtient des résultats corrects pour le fichier 8k qui descendent jusqu'à 40-50 pour des populations de 100 individus sur 100 générations.

En parallèle de l'implémentation de ces classes, 2 nouveaux fichiers ont été créés, l'un testant les méthodes de la classe des individus et l'autre testant les méthodes de la classe population. Pour ce MVP on arrive à une couverture de 84% du code car les fonctions de main et celles de traj3D ne sont pas testées, ainsi que les méthodes données dans le fichier original de la table des individus.

## 2 Amélioration du MVP : respect des contraintes, début d'amélioration de l'algorithme génétique et aspect utilisateur

Il y avait 3 objectifs d'amélioration clairs : Ajouter 2 nouvelles méthodes dans notre algorithme génétique, la sélection par tournoi qui pourrait remplacer celle par élitisme

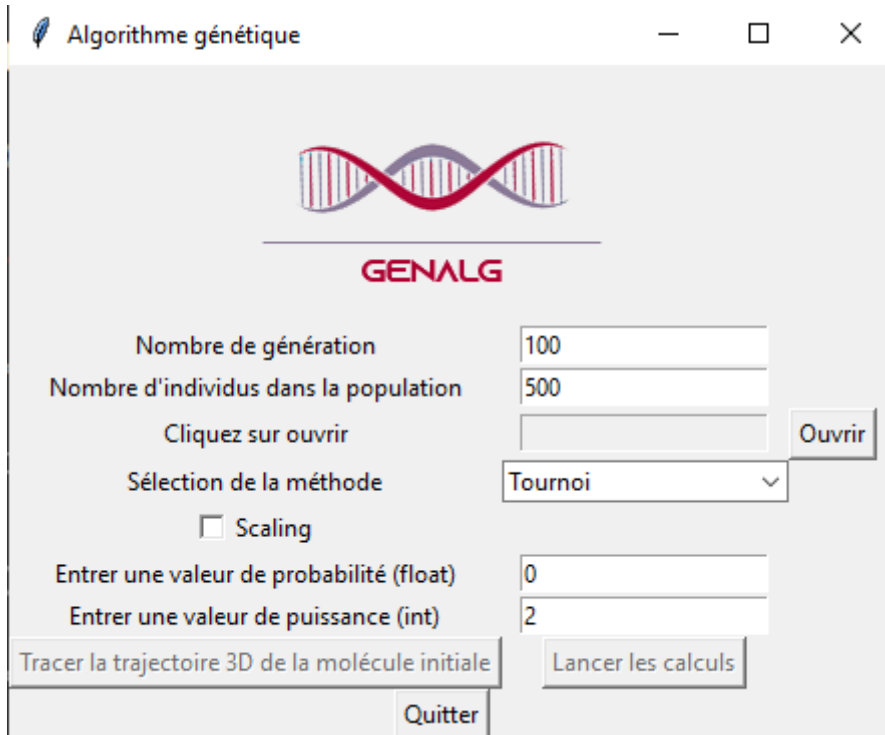
utilisée dans le MVP, un phénomène de mutation qui peut apparaître sur certains angles sous une loi de probabilité et une nouvelle fonction d'évaluation. Ensuite, rendre réaliste le résultat en respectant les contraintes de l'énoncé c'est à dire garder les angles à l'intérieur des bornes dans lesquelles ils sont compris et respecter la symétrie due à la réalité de ce qu'est un brin d'ADN. Et enfin la création d'un bouton qui s'affiche lors de la lancée de l'algorithme et qui permet de choisir le nombre de génération que l'on veut prendre, la séquence d'ADN en entrée et enfin permet d'afficher une représentation 3D du brin d'ADN à l'état final.

Pour le premier objectif, on a tout d'abord ajouté dans la classe population une méthode `selection_tournoi` qui compare deux à deux les individus de la population et ne garde que ceux qui "battent" les autres, à chaque fois ces 2 individus sont pris au hasard et on ne peut pas prendre 2 fois les mêmes. On ajoute une méthode `Mutate` dans la classe `RotTable` qui va prendre en entrée une valeur alpha qui va influencer sur la probabilité qu'un des angles d'une table de rotation change et la génération actuelle. Cette méthode, si l'angle change, va faire bouger l'angle selon une loi se rapprochant d'une loi normale dont le résultat sera additionné avec les valeurs d'angle actuelle mais, sa valeur est multipliée par un scalaire qui diminue au fil des générations et donc au fil des générations les mutations sont de plus en plus faibles. Et, on crée une nouvelle méthode d'évaluation qui prend également en compte l'angle que forme le dinucléotide initial et le dinucléotide final et donc avec une certaine pondération accorder une certaine importance à l'évaluation avec l'angle entre les angles rendu scalaire par un cosinus et une certaine importance à la distance euclidienne. On recherchera une optimisation de cette pondération dans un prochain temps. Ensuite, on pourra modifier la méthode d'évaluation avec du scaling qui lisse les résultats obtenus pour avoir moins de convergence locale en début de boucle. Enfin on a eu l'occasion d'implémenter une dernière méthode d'évaluation qui ne calcule plus un scalaire à partir de l'angle et la distance mais donne un couple (angle, distance) qui va favoriser la distance dans la sélection mais pour le croisement va optimiser l'angle pour mieux orienter les futures solutions.

Pour répondre au deuxième objectif, il a tout d'abord fallu changer la liste des clés que l'on utilisait et on a créé une nouvelle liste avec toutes les clés présentes dans la table de rotation initiale sauf celles où le symétrique de cette clé est déjà présente dans cette liste et on utilisait comme individu non plus une table de rotation avec 16 éléments mais une table de rotation avec 10 éléments. Il a ensuite fallu d'abord créer 3 nouvelles méthodes dans la classe des individus, la méthode `correspondance` qui associe à un nucléotide son symétrique, la méthode `symétrique` qui associe à un dinucléotide son symétrique en utilisant la méthode `correspondance` et la méthode `reconstitution` qui reconstruit la table de rotation à 16 éléments afin de pouvoir procéder à l'évaluation et qui sera à nouveau déconstruite après.

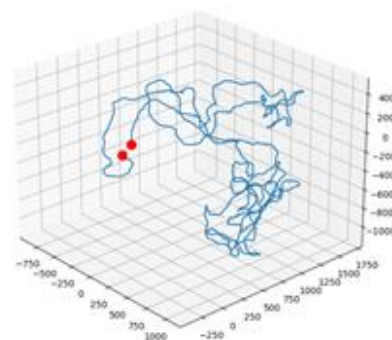
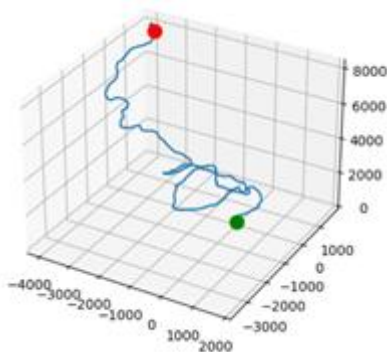
Le dernier objectif va pouvoir être réalisé grâce au module `tkinter` qui va permettre la création d'une interface graphique dans le lancement du programme. On a comme cité précédemment la possibilité de choisir le nombre de générations et le fichier d'entrée qui contiendra la séquence ainsi que la possibilité au début et la fin d'observer le brin d'ADN en 3D. Il y a également sur le bouton une jolie image pour rendre le bouton plus esthétique et

montre directement le thème du projet. Au fur et à mesure de l'avancée du projet, d'autres bouton sera rajouté pour pouvoir introduire les différentes méthodes dans les choix de l'utilisateur comme l'option avec scaling ou le choix de la valeur de probabilité pour le tournoi avec pondération du score.



Interface utilisateur finale

On rajoute également dans nos fichiers de tests les tests des méthodes implémentées dans cette partie et on arrive à une couverture de 85% du code.



On observe ci-dessus la différence entre le brin d'ADN avant et après qu'il a été soumis à l'algorithme génétique avec toutes les modifications mais sans l'optimisation des paramètres pour une population de 100 individus et 50 générations.

### 3) La dernière étape : optimiser les résultats et le temps de réponse de l'algorithme

Dans cette étape, le but est d'avoir un meilleur graphe que celui qu'on obtenait et plus rapidement qu'avant. Pour cela on procède à des méthodes d'optimisation des paramètres et des méthodes utilisées, et on tente de réduire la complexité temporelle de nos algorithmes.

Dans un premier temps, on utilise un algorithme de Benchmarking pour pouvoir comparer des paramètres d'entrer, pour tous les tests ci-dessous, on fera varier le nombre d'individus et le nombre de générations :

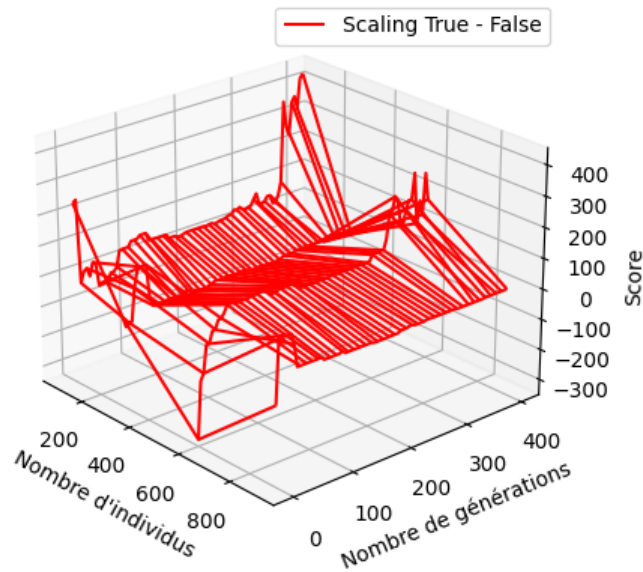
- On compare la sélection par tournoi et la sélection par élitisme, pour cette comparaison, on fait varier nombre d'individu entre 100 et 5100 avec un pas de 1000 et on fait varier le nombre de génération entre 10 et 510 avec un pas de 100.
- On compare les différentes sélections par tournoi avec la probabilité que les plus faibles gagnent, on fait varier le nombre d'individu de la même manière que le benchmark précédent. On fait ce benchmark dans les 2 cas avec et sans utiliser de scaling dans la méthode d'évaluation.
- La dernière comparaison est celle entre l'évaluation qui fait du scaling pour l'évaluation avec seulement la distance euclidienne qui est prise en compte et l'évaluation qui utilise la même méthode sans faire de scaling.

On optimise également le alpha correspondant à la probabilité qu'un certain individu mute. En utilisant le module `scipy.optimize` on trouve une valeur de alpha égale à 0.59 comme probabilité optimale qu'un individu mute.

### *III-Conclusion sur les résultats obtenus*

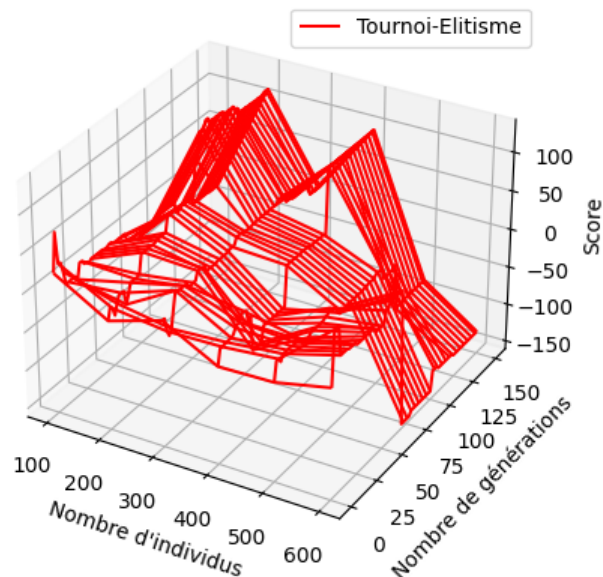
Finalement pour les 3 benchmarks qui ont été lancé, on a les résultats suivants.

## Surface Scaling - Non Scaling



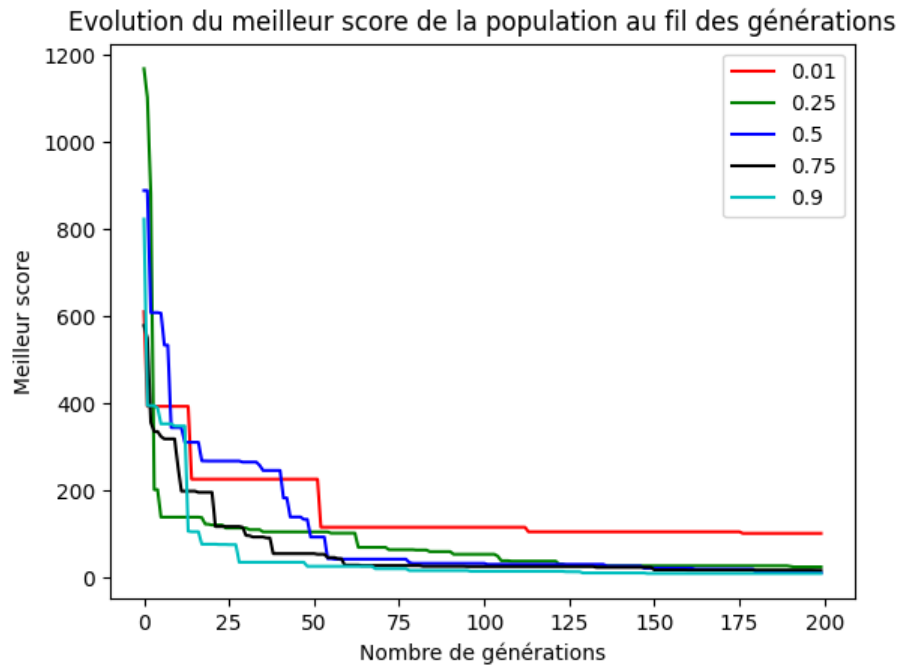
Tout d'abord on a le benchmark utilisé pour comparer la différence de score obtenue pour l'évaluation qui prend seulement en compte la distance euclidienne entre le premier et le dernier nucléotide de la chaîne. On a donc par le schéma ci-dessus une illustration de l'effet plutôt bénéfique qu'apporte le scaling sur la méthode d'évaluation.

## Comparaison de méthode de sélection : Elitisme vs Tournoi

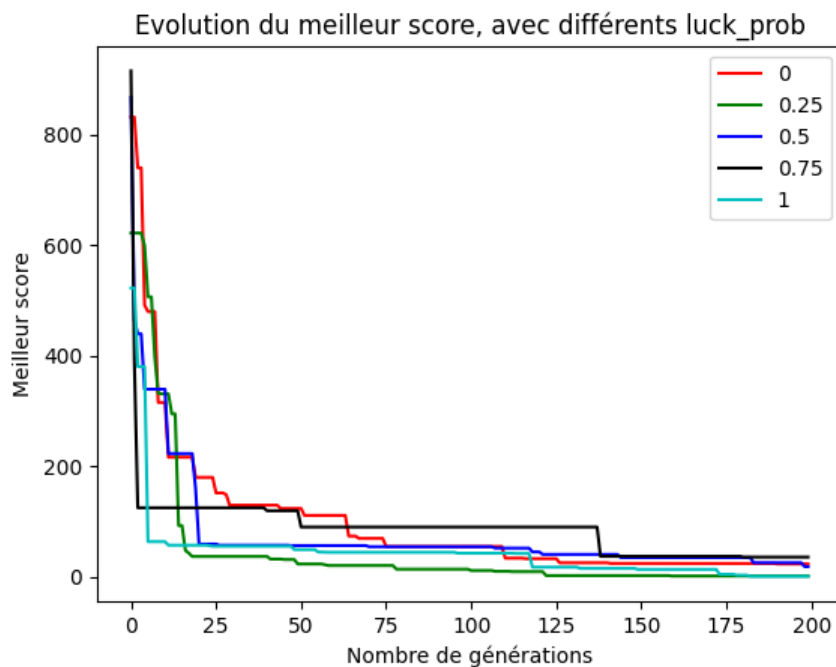


Ensuite vient la comparaison entre les 2 méthodes de sélection où sur le schéma ci-dessus on a la représentation de la différence de score obtenu entre ces 2 méthodes. On conclut par observation

graphique que la sélection par tournoi apporte de bien meilleures solutions quand le nombre d'individus et le nombre de génération deviennent grand.



Cas sans utiliser de scaling



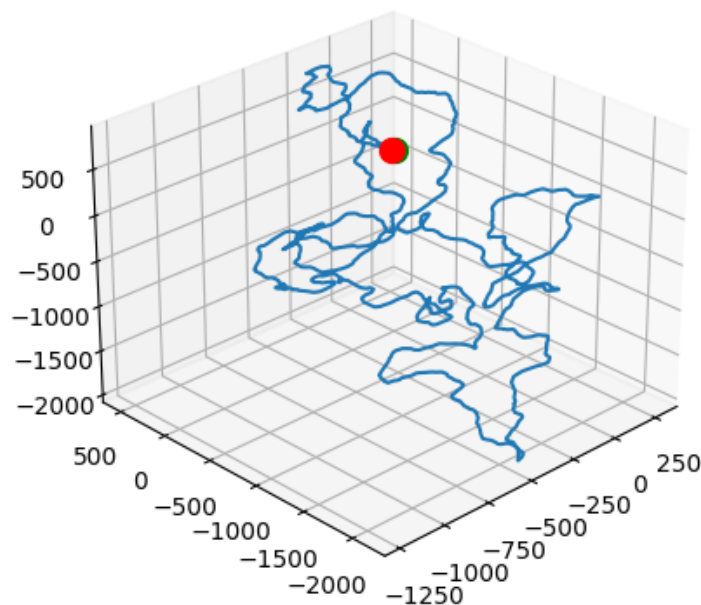
Cas avec utilisation de scaling

Enfin, dans ces derniers benchmarks, on observe graphiquement que la probabilité `luck_prob`, qui correspond à la probabilité en entrée de la méthode `tournoi_luck_prob` et qui donc donne une



chance aux plus faibles de battre les plus fort, ne doit pas être trop faible dans le cas où on n'utilise pas de scaling, on doit donc prendre au minimum une probabilité à 25% pour avoir des résultats optimaux en score. Pour le cas avec scaling, on n'observe pas de résultat qui soit plus optimal que d'autres.

Pour finir cette partie on va maintenant montrer le résultat final de notre projet avec les paramètres que nous avons tenté d'optimiser.



Voilà ci-dessus le résultat obtenu pour une population de 100 individus et pour 100 génération. Les méthodes utilisées ont été la sélection par tournoi et l'évaluation par la distance avec du scaling. On obtient une distance entre les 2 extrémités de 3, c'est ce qui a été cherché à optimiser dans cet exemple avec le fichier pasta 8k.

## Conclusion

Ce projet nous a permis de saisir les enjeux autour de l'algorithme génétique mais aussi de l'intérêt du biomimétisme afin de résoudre des problèmes d'optimisation. Nous avons développé un outil qui permet de résoudre le problème évoqué en un temps raisonnable et avec une qualité convenable.

Cependant, dans le temps imparti, nous n'avons pas pu mettre en place les stratégies d'optimisation qui auraient pu être pertinentes. Nous avons par exemple pensé, à l'utilisation du machine learning afin d'entraîner un modèle qui à partir d'un fichier fasta, détermine les

paramètres optimums pour rendre cyclique le fichier en question. D'autres part de nombreuses améliorations sont possibles en termes d'expérience utilisateur. Aussi, merci d'avoir lu ce rapport et merci pour cette semaine riche en travail et en nucléotides.

## Bibliographie :

- Towards data science, genetic algorithm implementation on python,  
<https://towardsdatascience.com/genetic-algorithm-implementation-in-python-5ab67bb124a6>
- Real-Coded Genetic Algorithms,  
<https://engineering.purdue.edu/~sudhoff/ee630/Lecture04.pdf>
- Algorithmes génétiques, Jean-Marc Alliot, Nicolas Durand,  
<http://pom.tls.cena.fr/GA/FAG/ag.html>