Rapport PI²

March 28, 2022

AGERT Petru-Luigi

DESPORTES Léonard

LACROIX Camille

MALCLES Héloïse

PIET Elliot

1 Contexte

1.1 Sujet

Ce projet s'inscrit dans un objectif scolaire. Nous allons travailler en partenariat avec l'entreprise Alten, par l'intermédiaire de Armand Leonardi.

L'entreprise Alten est une multinationale française d'ingénierie et conseil en technologies et services du numérique. Elle a été créée en 1988 par Simon Azoulay, Laurent Schwarz et Thierry Woog. Les activités ont été élargies jusqu'en 1997 puis l'entreprise a été introduite en bourse en 1999 et désormais est présente dans 28 pays. Alten propose à ses clients, par l'intermédiaire de consultants, des ingénieurs intervenant sur leurs projets technologiques.

Alten nous propose donc de composer un portefeuille par algorithme génétique. Pour y parvenir, nous avons récupéré le travail de nos camarades de l'année dernière. Nous avons eu accès à leur documentation ainsi qu'a leur code Python qui nous aidera nécessairement. Le but est de développer un prototype fonctionnel complet en utilisant un algorithme génétique afin de construire un portefeuille respectant les objectifs de rentabilité et de risque du client.

Pour débuter ce projet nous avons donc décidé de nous familiariser avec certaines notions telles que : la recherche heuristique, les algorithmes évolutionnaires, l'optimisation de Markovitz, modélisation de portefeuille d'actifs ...

L'objectif suivant sera donc de produire un outil complet avec une interface codées en Python. Pour cela, le script et les rapports de l'année précédente sont pris comme base pour être modifié et amélioré.

La méthode de Markovitz sera donc comparée à notre algorithme grâce à une étude approfondie des avantages et des inconvénients. Les résultats ainsi que les courbes et traitement de données permettront de comparer ces méthodes.

Le projet PI² s'inscrit dans une stratégie évolutive qui s'effectue en 5 ans. PI² signifie projet pour l'industrie et l'innovation et se déroulera du 29 Septembre 2021 au 01 Avril 2022.

Il s'appuie alors sur plusieurs piliers : des compétences techniques qui sont liées à nos majeures à l'ESILV (à savoir ingénierie financière et DIA) ; la consolidation des compétences de projet acquises lors des 3 premières années ; des méthodologies telles que l'innovation appliquée aux relations avec un partenaire extérieur ; des compétences douces notamment avec un profil MBTI pour l'équipe ; la valorisation du projet.

C'est un projet très valorisant pour les membres de l'équipe puisqu'il traite avec des partenaires extérieurs issus d'entreprises, il allie savoir, compétences, recherche et compétition.

Le projet s'effectue par équipe de 4 ou 5 personnes avec un total de 10h par semaine et par étudiant pour être équivalent à un ingénieur à temps plein.

Le rendu se fera évidemment à travers un résultat apportant une solution au problème posé mais également par un rapport avec tous les processus, étapes et résultats significatifs au cours du projet.

Des réunions de suivi avec notre partenaire sont faites régulièrement ainsi que des brainstormings avec l'équipe pour définir les objectifs actualisés et étudier les étapes/tâches à réaliser.

1.2 Définitions

1.2.1 Algorithmes génétiques

Les algorithmes génétiques s'appuient sur la sélection naturelle avec une population de solutions potentielles. Pour cela, on effectue des bonds successifs et on effectue une sélection qui peut se faire par plusieurs techniques :

Sélection par rang : On conserve uniquement les individus possédant les meilleurs scores d'adaptation. Le hasard ne fait pas partie du mode de sélection. On associe les individus à un rang en fonction de leur position (choisie grâce à l'évaluation). L'individu le moins bon sera associé au rang 1 et le meilleur au rang N pour une population de N individus.

Cette méthode présente des inconvénients comme lorsque la valeur d'évaluation des individus varie beaucoup mais également des avantages puisqu'elle permet de tester tous les individus qui ont alors une chance d'être sélectionnés. La convergence vers la bonne solution est plus lente mais efficace.

Sélection par tournoi : Il s'agit d'une technique utilisant la sélection proportionnelle sur des paires d'individus puis on choisit les meilleurs individus de chaque paire suivant le meilleur score d'adaptation. On converge donc ainsi vers une solution au problème.

Sélection uniforme : La sélection se fait de façon totalement aléatoire, uniforme et sans intervention de la valeur d'adaptation. En effet, chaque individu possède la même probabilité 1/P d'être sélectionné, pour une population de P individus.

Sélection steady-state : Le principe est de garder des populations améliorées à chaque génération en conservant une grande partie de la population précédente. A chaque génération, on sélectionne quelques individus parmi ceux qui ont les meilleures évaluations et on obtient ainsi une nouvelle population. On remplace ensuite les individus avec une évaluations plus faible. On garde donc ainsi les meilleurs individus de chaque génération.

Sélection élitiste : Cette sélection consiste à conserver les meilleurs individus pour la génération suivante. Ensuite, on génère le reste de la population selon l'algorithme de reproduction usuel. Cette méthode permet de ne pas perdre les meilleurs individus de chaque génération.

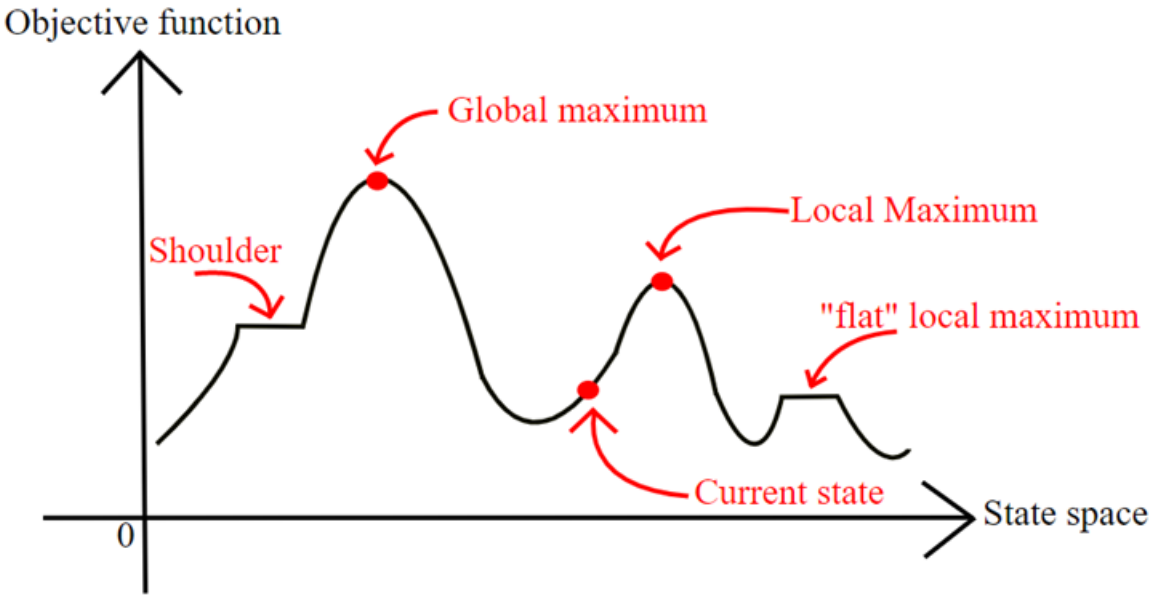


Figure 1 : Recherche des solutions d'un algorithme génétique

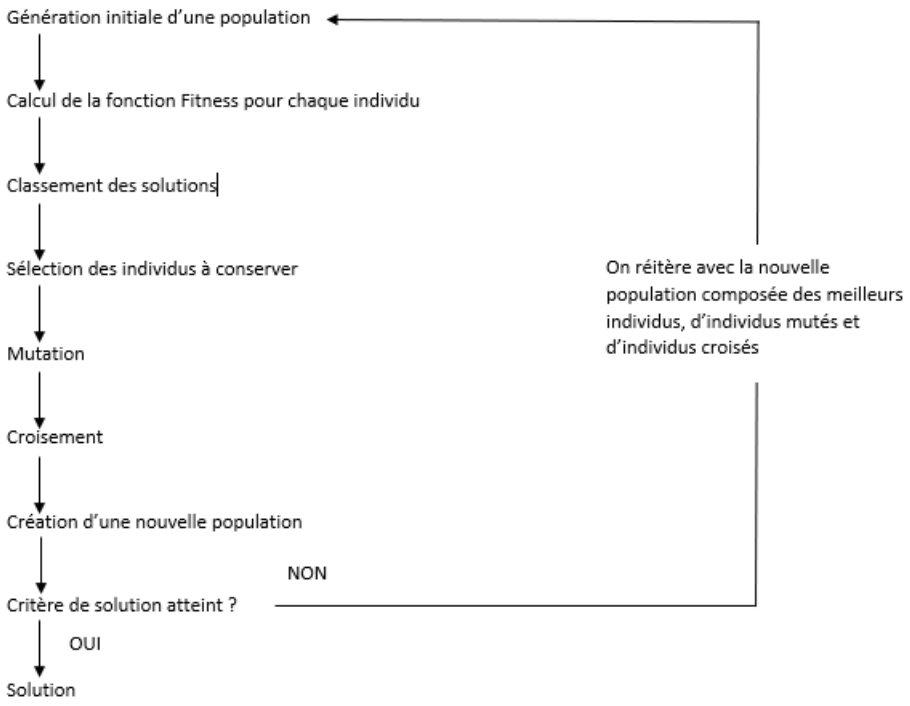


Figure 2 : Organigramme de l'algorithme génétique

1.2.2 Fonction fitness

La fonction fitness mesure la qualité de l'individu exprimée sous forme d'un nombre ou d'un vecteur. Il associe une valeur de performance à chaque individu, ce qui offre la possibilité de le comparer à d'autres individus. Cela permet également à l'algorithme génétique de déterminer qu'un individu sera sélectionné pour être reproduit ou pour déterminer s'il sera remplacé.

Il existe un impératif sur la valeur d'adaptation, il faut qu'elle soit croissante avec l'adaptation de la solution au problème. Un parcours valide renverra une valeur supérieure à un individu qui n'est pas solution au problème.

On peut également imposer une condition qui donne une fitness négative lorsqu'une solution n'est pas correcte et une fonction fitness positive lorsqu'elle répond au problème.

Pour choisir quelle solution sera la plus satisfaisante à ce problème, on étudie également le chemin qui sera le plus court. Ainsi, la fonction fitness qui lui sera associée sera importante. On a donc la fonction fitness qui varie dans le sens inverse de la distance correspondant au parcours.

1.2.3 Recherche heuristique

La recherche heuristique est une méthode de résolution de problème qui ne s'appuie pas sur une analyse détaillée ou exhaustive. Elle est à la base de beaucoup d'intelligence artificielle.

Il s'agit de s'approcher en testant des approches successives. Pour cela, il faut s'appuyer sur des similitudes et garder une série limitée de solutions pour tendre vers la solution optimale. Pour cela on utilise une fonction appelée heuristique h(n) qui estime le coût du chemin le plus court pour se rendre à l'objectif.

Une heuristique est utilisée pour guider la recherche : u les heuristiques exploitent les connaissances du domaine d'application.

Il existe plusieurs types de recherches :

Recherche du meilleur d'abord : Il s'agit d'une stratégie définie en choisissant un ordre dans lequel les états sont développés. Pour cela, on utilise une fonction heuristique et on mesure l'utilité d'un nœud

Recherche gloutonne : Il s'agit d'une fonction d'évaluation f(n) = h(n). On développe le nœud qui paraît le plus proche de l'état final. Toutefois il existe un problème puisqu'il y a un risque de rester bloquer dans des boucles sauf si on ajoute un test pour éviter les répétitions.

Recherche A* : l'algorithme A* est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final tous deux donnés. Il utilise une évaluation heuristique sur chaque nœud pour estimer le meilleur chemin y passant, et visite ensuite les nœuds par ordre de cette évaluation heuristique. Cette recherche permet d'éviter de développer des chemins déjà chers en utilisant une fonction f(n) = h(n) + g(n).

F(n) : Cout total

H(n) : Cout pour atteindre l'état final

G(n) : Cout de l'état initial – l'état n

Recherche locale : Dans certains problèmes, le chemin n'a pas d'importance. L'état lui-même est la solution et suffit à répondre au problème. Pour cela on teste l'ensemble des possibilités et celle qui sont solutions sont conservées quel que soit le chemin emprunté.

1.2.4 Optimisation de Markovitz

La théorie moderne du portefeuille est une théorie financière développée en 1952 par Harry Markowitz.

Harry Markowitz est un professeur qui a reçu un prix Nobel d'économie en 1990 pour avoir développé la théorie dite du choix de portefeuilles pour le placement des fortunes. Les travaux de Markowitz ont constitué la première tentative de théorisation de la gestion financière et son modèle suggère une procédure de sélection des titres boursiers, à partir de critères statistiques afin d'obtenir des portefeuilles optimaux.

Cette théorie utilise la diversification afin d'optimiser le portefeuille d'investisseurs. Elle fait appel aux concepts de frontière efficiente et au prix d'un actif étant donné son risque par rapport au risque moyen du marché.

Pour cela on pose plusieurs hypothèses :

Le comportement de tous les investisseurs est caractérisé par un degré plus ou moins prononcé d'aversion vis-à-vis du risque. Ce dernier est mesuré par l'écart-type de la distribution de probabilité du rendement.

Les investisseurs son rationnels : bien que leur fonction de préférence soit purement subjective, ils opèrent, en référence à celle-ci, des choix strictement transitifs.

Tous les investisseurs ont le même horizon de décision, qui comporte une seule période. Cette simplification, qui peut paraître exagérée, permet de mettre en œuvre un modèle de décision qui tient compte du caractère hautement combinatoire du portefeuille.

La théorie financière met l'accent dans le cadre de la gestion de portefeuille sur deux critères essentiels, à savoir, la rentabilité et le risque.

La maximisation de rentabilité exige une gestion minutieuse des risques et cette dernière se fait à base de la diversification. La diversification stipule le mixage d'un portefeuille d'actifs entre ceux risqués ou bien les combiner avec d'autres sans risque. C'est l'investissement dans différentes classes d'actifs ou dans différents secteurs, cette diversification ne signifie pas seulement détenir beaucoup d'actifs.

Grâce à ce modèle, le rendement d'un portefeuille consiste en la somme des rendements des actifs qui le composent, pondérés par leur poids. Le rendement est alors :

$\mathrm{E}(\mathrm{R}_p) = \sum w_i E(R_r)$

Le risque est défini par la volatilité du portefeuille correspond à son écart-type. Pour un portefeuille composé de 2 actifs on a :

$\sigma_p^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j \sigma_{ij} = \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j \sigma_i \sigma_j \rho_{ij}$

Pour représenter le rendement de chaque titre dans le portefeuille, on utilise le rendement suivant le niveau de risque. Une représentation de la frontière efficiente sur laquelle se situe les portefeuilles composés de titres individuels offrant le meilleur rendement pour un certain niveau de risque permet de visualiser la solution optimale.

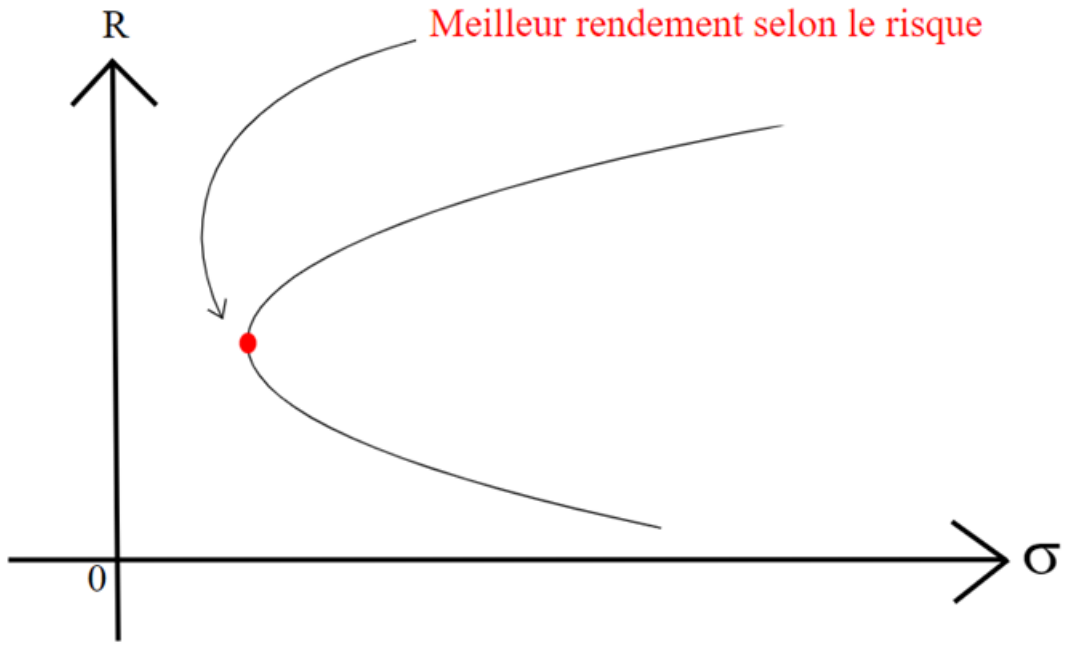


Figure 3 : Frontière efficiente du rendement selon le risque

1.2.5 Interface Homme Machine (IHM)

L'interaction Homme-Machine (IHM) traite de la conception et du développement de systèmes interactifs en tenant compte des impacts sociétaux et éthiques. L'IHM fait référence à un tableau de bord qui permet à un utilisateur de communiquer avec une machine, un programme informatique ou un système.

Ce système peut être appliqué à n'importe quel écran pour interagir avec un appareil, mais est généralement employé pour décrire des écrans utilisés dans les environnements industriels. Cette interface affiche les données en temps réel et permettent à l'utilisateur de contrôler les machines à l'aide d'une interface utilisateur graphique.

La facilitation de l'utilisation de dispositifs augmente avec le nombre d'interfaces numériques dans la vie quotidienne. L'IHM a pour objectif d'être le plus efficace, accessible et intuitif possible pour les utilisateurs de compléter une tache au mieux. On utilise la linguistique, la vision par ordinateur et l'humain pour se faire.

Dans un environnement industriel, une interface peut prendre plusieurs formes que ce soit un écran autonome, un tableau de bord attaché à un autre équipement ou encore une tablette. Cela permet aux utilisateurs de visualiser les données concernant les opérations et le contrôle de machines.

1.3 Environnements

1.3.1 MySQL

Le nom vient de l'association de « My », le nom de la fille du co-fondateur, et de SQL qui signifie Structured Query Language, qui est un langage de programmation qui vous aide à accéder et gérer les données dans une base de données relationnelle. Ce modèle de gestion de données est en concurrence avec Oracle, PostgreSQL et Microsoft SQL Server.

MySQL est un système de gestion de bases de données relationnelles (SGBDR), distribué sous une double licence GPL et propriétaire. Ce logiciel est utilisé à la fois par les particuliers et les professionnels.

Une base de données est une collection structurée de données, organisée pour faciliter l'utilisation et la récupération. Une base de données permet de stocker et de retrouver des données structurées,

semi-structurées ou des données brutes ou de l'information, souvent en rapport avec un thème ou une activité ; celles-ci peuvent être de natures différentes et plus ou moins reliées entre elles.

MySQL est un système populaire qui peut stocker et gérer ces données pour l'utilisateur grâce à une approche appelée base de données relationnelle. On divise les données en plusieurs zones de stockage séparées, appelées tables, plutôt que de tout regrouper dans une seule grande unité de stockage.

1.3.2 Python

Python est un langage de programmation open source créé par le programmeur Guido van Rossum en 1991. Il tire son nom de l'émission Monty Python's Flying Circus. Il s'agit d'un langage de programmation interprété, qui ne nécessite donc pas d'être compilé pour fonctionner. Il favorise la programmation structurée, fonctionnelle et orientée objet. Ce langage utilise un typage dynamique fort, d'une gestion automatique de la mémoire et d'un système de gestion d'exceptions.

Python est disponible sur toutes les plateformes informatiques, à savoir sur les smartphones et ordinateurs centraux sous Windows, Linuw, MacOS, Androidn iOS et Unix. Ce langage peut également être traduti en Java ou .NET.

Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser.

Il existe toutefois un inconvénient, la rapidité est moindre comparé aux langages compilés comme le C. On l'appelle langage de programmation de haut niveau. Il s'agit d'un langage idéal pour les débutants et confirmés. Python permet de se focaliser sur ce que les utilisateurs font plutôt que sur la façon. Cela rend la programmation plus rapide et plus simple à apprendre et utiliser.

Les caractéristiques sont donc peu nombreuses, ce qui permet de créer des programmes avec peu d'efforts. La syntaxe est lisible et directe. Ce langage fonctionne sur tous les principaux systèmes d'exploitation et plateformes informatiques. Malgré sa lenteur, ce langage permet de créer des logiciels de qualité professionnelle, pour les applications et services Web.

1.3.3 Programmation orientée objet

La programmation orientée objet est une représentation de programmation informatique. Il s'agit de la définition et de l'interaction des logiciels, appelés objets (concept, idée et entité du monde physique). Il faut représenter les objets et leurs relations qui permettent de concevoir et réaliser les fonctionnalités attendues, de résoudre les problèmes. Un objet peut être défini comme un champ de données avec des attributs et comportement unique.

Il s'agit d'abord de collecter tous les objets que le programmeur souhaite manipuler. Il faut pour cela identifier les liens et connexion entre ces objets. C'est ce qu'on appelle le « Data Modeling » (modélisation des données). L'étape de modélisation est importante et nécessite la POO pour transcrire les éléments réels sous forme virtuelle. On utilise donc pour cela des techniques de programmation pour mettre en œuvre une conception basée sur les objets et des méthodologies de développement logiciel objet (telles que le processus unifié : « Unified Software Development Process ») et exprimée à l'aide de langages de modélisation (tels que UML : « Unified Modeling Language »).

Une fois l'objet connu, il est étiqueté avec une classe objets définissant le type dedonnées qu'il contient ainsi que les séquences logiques (appelées « méthodes ») permettant de le manipuler. Les objets peuvent communiquer par le biais d'interfaces appelées « messages ». La programmation orientée objet peut se faire par plusieurs langages de programmation à savoir Python, Java, Javascript, C++, Visual Basic.NET, Ruby, Scala et PHP.

Les principes clés sur lesquels reposent la POO sont :

- L'encapsulation : la catégorisation de chaque objet en classe d'objet spécifique. Les objets qui n'ont pas la même classe n'y ont pas accès et ne peuvent effectuer de modifications. Ils peuvent uniquement faire appel à une liste de fonctions ou méthodes publiques. Ce principe permet la sécurisation et évité la corruption des données.

- L'abstraction : les objets révèlent les mécanismes internes pertinents pour l'utilisation d'autres objets, dissimulant les codes d'implémentation inutile. Les changements et ajouts sont donc plus facile au fil du temps grâce aux développeurs.

- L'héritage : permet d'ajouter les relations et sous-classes entre les objets. Les développeurs peuvent réutiliser une logique en commun sans changer la hiérarchie. L'analyse est donc plus complète, le temps de développement considérablement réduit et la précision augmentée.

- Le polymorphisme : la dernière spécificité de la programmation orientée objet. Les objets peuvent prendre différentes formes en fonction du contexte. Le code n'a pas besoin d'être dupliquer puisque le programme détermine automatiquement quel usage est requis pour chaque exécution du même objet.

1.3.4 LateX

LaTeX est un système logiciel de composition de documents, une collection de macro-commandes destinées à faciliter l'utilisation du « processeur de texte. Ce terme vient du créateur du logiciel appelé Leslie Lamport, c'est donc une abréviation de Lamport TeX. Il a été développé au début des années 1980 et est maintenu par une équipe de bénévoles depuis 1989.

Il s'agit d'un langage et système de composition de documents. Il s'agit d'une collection de macro-commandes destinées à faciliter l'utilisation du processeur de texte. LaTeX permet de rédiger des documents dont la mise en page est réalisée automatiquement en se conformant du mieux possible à des normes typographiques. Une particularité de ce logiciel est son mode mathématique qui permet de composer des formules complexes. Il est devenu la méthode privilégiée d'écriture de documents scientifiques de tailles variées (articles, thèses ...) ainsi que de différents types (lettres, transparents ...).

L'évolution de LaTeX est assurée par une communauté active et structurée en groupes d'utilisateurs. Ces qualités en font l'outil de rédaction privilégié des mondes universitaires et scientifiques dans certaines disciplines.

Ce logiciel est particulièrement utilisé dans les domaines techniques et scientifiques pour la production de documents. Il peut également être employé pour générer des documents de types très variés. De nombreux sites utilisent ce sous-ensemble LaTeX pour composer notamment leurs formules mathématiques.

LaTeX demande de se concentrer sur la structure logique du document et la mise en page est gérée par le logiciel. Cela permet la séparation de la forme du contenu contrairement à certains logiciels tels que OpenOffice.org Writer ou encore Microsoft Word. LaTeX requiert, en revanche, un apprentissage initial plus important que celui nécessaire pour les autres logiciels de composition de documents. La qualité du document produit est élevée (formules mathématiques, respect des règles typographiques), la gestion des références bibliographiques, les numérotations et table de matières sont cohérentes.

1.3.5 Bloomberg

Bloomberg LP est un groupe financier américain financier spécialisé dans les services aux professionnels des marchés financiers et dans l'information économique et financière. Bloomberg agit aussi bien en tant qu'agence de presse qu'en tant que média (télévision, radio, presse, internet et livres). Ce groupe fournit des actualités et des données financières en temps réel.

Cette entreprise est créée en 1981 par Michael Bloomberg, qui était maire de New York de 2002 à 2013. L'entreprise est actuellement composée de 15500 employés répartis dans 130 pays. Les principaux concurrents sont SIX Financial Information, Thomson Reuters, FactSet Research Systems, Dow Jones....

Le groupe a débuté avec le rachat à la Banque d'investissement Salomon Brothers d'une base de données historiques sur les courbes de taux des emprunts d'État du Trésor américain. Bloomberg a connu un véritable succès en plaçant sur l'ensemble de ses terminaux un système de messagerie entre professionnels, bien avant l'ère d'Internet. Ces terminaux multifonctions ont également été enrichis de l'actualité du secteur économique et financier et des retransmissions de cours d'instruments financiers.

L'utilisation de Bloomberg ce fait grâce à des terminaux spécifiques. Le fameux terminal Bloomberg constitue un de leurs produits star. Il s'agit d'une plateforme intégrée qui rassemble et diffuse les données de prix, les données comptables et financières des entreprises, les statistiques des transactions des marchés ainsi que les actualités financières.

Le terminal Bloomberg dispose de très nombreux outils permettant l'analyse financière, l'analyse fondamentale ou l'analyse technique. Bloomberg est également une des principales sources d'information

qualitative des professionnels de la finance et permettent également aux étudiants d'entrer directement en contact avec des professionnels de la finance.

1.4 Composition de portefeuille

1.4.1 Prix d'un actif

On parle de portefeuille d'actifs mais qu'est-ce qu'un actif ? Un actif financier est un titre ou un contrat qui est susceptible de produire à son détenteur des bénéfices ou des pertes en capital, en contrepartie d'une certaine prise de risque. Les actifs sont généralement transmissibles et négociables (notamment sur le marché financier).

Les placements financiers effectués par une entreprise représentent généralement des actifs financiers. Ils peuvent être détenus à court terme ou à long terme. C'est un bien immatériel et qui revêt une nature monétaire. Ces actifs financiers comprennent deux types d'actifs : Les immobilisations financières (actions, créances, prêts, cautions, dépôts de garantie) Les valeurs mobilières (sociétés d'investissement à capital variable, fonds communs de placement)

Les actifs financiers se répartissent principalement entre les titres de créance ou de dette, c'est-à-dire les obligations et assimilés, et les titres de capital, qui correspondent aux actions et autres participations.*

Il existe par ailleurs plusieurs types d'actifs :

- Les actifs sous-jacents : ils comprennent à la fois les actions, les indices, les matières premières, les obligations et les ETP. Ce sont tous les produits financiers qui définissent le cours des produits dérivés et ainsi le profit ou la perte résultant d'une opération sur ce type de produits. Lors d'un investissement sur les produits dérivés, on ne possède pas l'actif sous-jacent. La transaction engendrera un profit si la prédiction suit une évolution à la hausse de l'actif et inversement.

- Les actifs de société : ce sont les ressources qui ajoutent de la valeur à l'entreprise ou qui pourraient générer de la valeur dans le futur. La valeur d'une entreprise s'apprécie à partir de son bilan financier qui détermine comment elle gère ses actifs, et qui inclut les actifs courants tels que les liquidités ou les stocks et les actifs immobilisés sur une période de plus d'un an comme l'immobilier et l'équipement.

1.4.2 Portefeuille d'actifs

Un portefeuille d'actifs désigne une collection d'actifs financiers détenus par un établissement ou un individu. Cela peut aussi désigner des valeurs mobilières détenues à titre d'investissements, de dépôt, de provision ou de garantie. Les portefeuilles sont caractérisés par le degré de diversification qui permet d'atteindre un juste milieu entre le risque, la volatilité et la rentabilité du portefeuille, tout en tenant compte de la durée prévue du placement. Il existe d'ailleurs plusieurs types de gestion :

- Gestion conseillée : le détenteur du portefeuille continue à piloter lui-même son portefeuille en sollicitant, si besoin est, les conseils d'un spécialiste.

- Gestion pilotée : consiste à confier à un professionnel la gestion de tout ou partie d'un portefeuille, selon une orientation déterminée préalablement (défensive, offensive, etc.). Le gérant peut prendre des décisions d'achat ou de vente en fonction des objectifs fixés par un investisseur.

- Gestion sous mandat : où un gérant agréé par l'AMF prend en charge la gestion complète du portefeuille (moyennant rémunération) en respectant le couple rendement/risque fixé par le client.

1.4.3 Risque

Un risque financier est un risque de perdre de l'argent à a suite d'une opération financière ou à une opération économique ayant une incidence financière. Une entreprise cherche à limiter son exposition aux risques financiers afin de sécuriser ses liquidités, préserver sa performance financière et réduire les impacts de la volatilité sur son résultat.

Ce risque comprend différents types de risques :

Risque du marché : Le risque de marché est un risque global de perte financière lié à la variation des cours de tous les produits qui composent un portefeuille. Cela comprend : le risque de taux (ou

taux d'intérêt, qui est le risque financier qu'un produit perde de la valeur à la suite d'une diminution ou d'une augmentation des taux d'intérêts) ; le risque de change (risque financier de voir son investissement perdre de la valeur à cause d'une variation des taux de change) ; le risque action (qui correspond à la possibilité de subir une perte de capital entre le moment d'achat de l'actif et le moment de sa revente) ; le risque matière première (qui impacte directement les entreprises qui dépendent de la production et la transformation de matières premières et d'énergies).

Risque de crédit/contrepartie : Ce risque traduit la possibilité que la qualité de remboursement ou la capacité à contracter un crédit par l'entreprise soit réduite. Il s'agit du risque financier que la qualité de remboursement de l'emprunteur soit réduite. Cela peut engendrer une baisse de la valeur des titres de créance de l'entreprise.

Risque de liquidité : il s'agit d'un risque de ne pas pouvoir revendre ses titres du fait d'un manque de volume des transactions. Ce risque est une situation pouvant naître du fait qu'une entreprise ou un détenteur de portefeuille ne puisse pas revendre ses titres. Les risques financiers de liquidité peuvent affecter les portefeuilles par manque de volume de transactions. La conséquence directe est que souvent, les entreprises se voient dans l'obligation de brader les titres ou les marchandises pour obtenir de la liquidité. On parle de marché liquide lorsque le volume des transactions est assez élevé pour pouvoir vendre ses titres sans difficulté.

Risque opérationnel : il correspond aux pertes éventuelles de l'entreprise provoquées par des erreurs du personnel ou des défauts commis par les ressours humaines ou les machines de production : défaillances des logiciels, fraudes de certains agents, erreur de frappe ...

Risque exogène : Parmi les risques n'ayant pas été cité précédemment il existe de nombreux risques extérieurs tels que la politique, la réglementation, la météo, le pays...

Le risque politique correspond aux risques que les décisions ou évènements à caractère politique aient pour conséquence d'entraîner des pertes financières (pour les entreprises ou les particuliers). Une crise politique peut engendrer l'exclusion de l'Etat par la communauté internationale de certains marchés.

Le risque réglementaire est similaire au risque politique. L'adoption d'une loi ou le changement d'un régime juridique des suites d'une nouvelle réglementation peut emporter des risques financiers pour les entreprises exerçant dans les secteurs concernés. Le risque météo est le risque de perte potentielle de chiffre d'affaires ou de profit due aux variations de la météo (température, précipitations, ensoleillement, vent).

1.4.4 Diversification

La diversification est le fait d'investir dans plusieurs classes d'actifs afin de réduire l'exposition du capital à un actif ou risque particulier. Il s'agit d'une stratégie d'investissement qui diversifie les actifs contenus dans un portefeuille, soit qui répartit les capitaux entre les différentes classes d'actifs monétaires et financiers. Cela signifie que cette stratégie vise à trouver un équilibre entre le rendement et le risque. Le but est de mieux protéger les capitaux placés des risques liés à une détention d'un nombre limité d'actifs.

Une diversification permet en cas de crise économique de compenser certains investissements atteints par la crise et ceux qui ne seront pas touché, voire progresseront. Elle permet également de dégager des rendements globaux plus réguliers et moins volatils qu'en investissant sur une seule classe d'actifs ou une seule entreprise.

Il existe plusieurs facteurs de diversification de son allocation : le nombre de titres dans lesquels on choisit d'investir, la classe des actifs choisis, mais également les zones géographiques ou les secteurs.

Un nombre important de titres permet de réduire le risque de perte en capital et la volatilité de son portefeuille. En effet, en investissant dans un seul titre, si l'entreprise fait faillite, vous perdez l'intégralité de vos investissements.

La classe de l'actif est aussi un facteur important de la diversification de portefeuille. C'est-à-dire qu'il est nécessaire de ne pas choisir uniquement des actions ou uniquement des obligations. En temps de récession, les actions vont généralement perdre de leur valeur. Les investisseurs ont tendance à transférer leurs investissements vers des actifs moins risqués, en particulier les obligations et notamment les obligations d'Etat, de sorte que la valeur de cette classe d'actif s'apprécie. Les pertes des unes seront compensées par les gains des autres.

Un autre facteur important est la diversification des zones géographiques. En effet, toutes les régions n'ont pas les mêmes enjeux économiques et toutes ne répondent pas aux mêmes changements politiques et géopolitiques. Ainsi, il est recommandé d'avoir un portefeuille de titres internationaux.

Une diversification sectorielle est également importante. Des entreprises dans des secteurs d'activité différents auront moins tendance à être corrélées, cela permet ainsi de limiter le risque. Il n'est donc pas recommandé de ne détenir que des titres d'un même secteur économique.

2 Organisation du travail

2.1 Outils

Au niveau de l'organisation du travail, on a travaillé sur plusieurs plateformes, logiciels et langages. Nous avons travaillé via VS Code ce qui était pour la plupart la première fois et qui nous permet d'utiliser plusieurs langages sur la même plateforme. Nous avons créé la base de données sur MySQL que nous avions récupérée de Bloomberg. Nous avons donc codé sur Python qui était un langage connu pour nous.

Pour communiquer nous avons utilisé Messenger qui est un canal facile pour parler de nos avancements respectifs. Nous essayions de communiquer en continu pour ne pas travailler sur les mêmes choses et se compléter. Pour partager nos fichiers nous avons utilisé Teams qui nous permettait de modifier nos fichiers en temps réel et à plusieurs et de regrouper tous les dossiers nécessaires à l'avancement et aux rendus. Pour le code, nous avons fait la mise en commun du code sur Github. Nous avons créé un repository commun qui permet à tout le monde de push et pull le code sur des branches respectives. A la fin nous avons tout mis sur le master qui est la version finale du projet. Enfin, nous avons fait des réunions régulièrement au sein de l'équipe sur Zoom pour partager nos écrans et montrer nos avancées. Les réunions avec notre intervenant se faisaient majoritairement en présentiel au pôle une semaine sur deux avec l'autre équipe partenaire. Nous faisions un point sur nos avancées respectives ainsi que sur les prochains objectifs à atteindre et à développer. C'était également le moment de poser nos questions et d'échanger sur les points de désaccord entre équipe ainsi qu'avec notre partenaire.

2.2 Organisation globale

Pour effectuer au mieux ce projet nous avons débuté par une grosse phase de documentation et de recherches pour comprendre les concepts que nous allions aborder, les thèmes financiers et informatiques. Nous avons donc répertorié des définitions et notions dans ce rapport ce qui nous a permis de ne pas buter sur des concepts lors de la réalisation de l'algorithme. Nous avons également récupéré le travail de l'équipe de l'année précédente que nous avons étudié et que nous avons réutilisé. Nous avons ensuite mis en place l'algorithme et le rapport en parallèle pour ne pas prendre de retard. Donc au fur et à mesure de l'avancement nous avons noté nos problèmes ainsi que nos modifications.

Des livrables nous ont été réclamés notamment au début et à la fin du projet. Au début pour présenter le projet et voir si nous avions connaissance des objectifs. Nous avons donc réalisé un pitch vidéo, des réunions régulières (Réunion pour créer le profil MBTI de l'équipe, réunion personnelle avec un facilitateur pour avoir un retour d'expérience et un avis sur le projet de chacun, réunion avec un auditeur en fin de parcours pour évaluer et conclure le projet), un poster à créer pour le showroom final ainsi que des pitch et vidéos pour ce même évènement. Nous avons également dû écrire un rapport en anglais pour les livrables ainsi qu'un abstract en français et un en anglais. Enfin une vidéo de 2 minutes pour résumer le projet nous a été demandée ainsi que nos livrables de sorties pour l'intervenant, à savoir le code Python et le rapport.

2.3 Répartition des tâches

Concernant la répartition des tâches, au cours des réunions avec ou sans le professionnel d'Alten, nous planifions les futures tâches à accomplir dans les jours à venir. La répartition des tâches n'était pas toujours définie. Le rapport a dès le début été pris en main par Camille qui géré les recherches. Elliot a rapidement récupéré les données sur Bloomberg pour créer la base de données avec Héloïse. Léonard s'est penché sur le code et la récupération de l'année précédente.

L'étape suivante a été de commencer à mettre en place les classes. Héloïse et Léonard ont commencé par établir la connexion entre la base de données et Python pour récupérer les données et les utiliser dans les classes. Nous avons donc pour ce récupéré le code des années précédentes qui marchait très bien pour la connexion.

En parallèle, Camille a créé un Github pour regrouper le travail de tous. Concernant les livrables, des réunions étaient faite pour que tout le monde soit d'accord sur les rendus à faire.

Le code a été avancé un peu par tout le monde par la suite avec une grosse contribution de Léonard. Certains problèmes ont été rencontré et Héloïse et Camille se sont régulièrement appelées pour avancer dessus et régler les modifications à faire tout en avançant le rapport en parallèle.

Tout au long du projet, les tâches étaient mises à jour et nous communiquions beaucoup pour ne pas travailler sur les mêmes choses et se faire part de nos avancées. Messenger était le canal idéal. Nous avons beaucoup discuté avec l'équipe partenaire concernant de points de divergence, notamment pour le calcul de rendements. Il est important d'avoir un avis autre et cela nous a permis d'aboutir à quelque chose de cohérent.

Il s'est ensuite agit de rentrer dans le vif du sujet à savoir la partie algorithme génétique, mutation et croisement. Héloïse et Camille ont travaillé sur le calcul de score ainsi que les calculs de rendements avec les périodes de rebalancement. Petru-Luigi a essayé de trouver des moyens d'améliorer la fonction de fitness de façon théorique en proposant des pénalités. Léonard s'est occupé de faire les croisements et mutations et Elliot a terminé l'interface graphique entamée par Camille.

Quant aux derniers livrables, nous nous sommes réparti les tâches. Camille qui avait géré le rapport s'est donc chargé de faire les abstracts en français et en anglais ainsi que du rapport en français pour l'entreprise et en anglais pour l'école. Le poster que l'on devait préparer pour le showroom a été fait par Héloïse avec l'aide d'Elliot qui a fait un logo. Pour la vidéo de 2 minutes de présentation nous nous sommes concertés et avons décidé de faire un résumé de notre projet avec un audio en fond et des images du code qui montrent comment l'utilisateur rentreraient ses données et ce que lui renverrait notre algorithme. Il fallait également faire un visuel différent du poster dont s'est chargé Elliot.

2.4 Gestion du temps, deadlines, risque

Au niveau de la gestion du temps, nous avons commencé les recherches dès que les sujets nous ont été attribués. Nous avons rapidement eu une réunion ou Armand Leonardi nous a expliqué ses attentes et l'objectif du projet. Rapidement nous avons démarré les recherches et la répartition des tâches. Les réunions toutes les 2 semaines nous imposaient une deadline ce qui nous obligeait à travailler et donc nous donnaient un certain rythme. Nous avions donc une moyenne de 6-7h par personne et par semaine de travail sur ce projet pour égaler un ingénieur à temps plein qui serait sur 35h de travail. En réalité, nos emplois du temps sont très inégaux. Certaines semaines nous passions chacun 10h sur le projet et d'autres seulement 3h car les cours étaient trop prenants.

Cette année nous avions peu de deadlines en comparaison aux autres années. Il n'y avait aucune deadline durant l'année à l'exception de la fin de du deuxième semestre. Il y a eu toutes les rendus et rendez-vous d'un coup à la fin pour conclure le projet les jours d'avant le showroom.

Le sujet étant complexe il laissait place à l'interprétation intellectuelle donc il a fallu parfois faire attention à ne pas trop dévier du sujet et de la trajectoire initiale et s'assurer d'avoir une partie fonctionnelle avant d'améliorer notre fonction fitness. Les réunions régulières nous remettaient également dans le droit chemin ainsi que les discussions avec l'équipe partenaire.

3 Développement

3.1 Algorithmes

3.1.1 Données Bloomberg

La première étape lors de la création de l'algorithme a été de créé la liaison avec la base de données créée à l'aide des données Bloomberg. Pour cela, on récupère les données Bloomberg que l'on place dans un .csv.

On créé une base de données SQL qui récupère ces données en leur attribuant un nom, une date, une valeur, un rendement et un volume.

3.1.2 Classe Connexion

On créé ensuite une première classe 'Connexion'. Cette classe permet de récupérer les données de la base de données grâce à la fonction pymysql.cursors. La fonction 'Initialisation' ouvre la connexion avec la base de données.

On créé une fonction 'Execute' qui prend en argument une requête SQL sous forme d'une chaîne de caractères et retourne le curseur qui permet d'accéder au résultat de la requête. Enfin, on ferme la connexion.

3.1.3 Classe Actifs

La seconde classe est la classe 'Actifs'. Pour cela, on initialise les actifs avec un nom, une valeur, un volume, une date, un nombre de shares, un rendement, un poids et une liste de rendements et valeurs des actifs entre 2 dates. On créé une fonction 'creationActifs' qui prend en argument la connexion avec la base de données et qui retourne une liste d'actifs avec le nom de chaque actif et les autres attributs sont initialisés à 0.

Pour cela, on utilise la requête "Select distinct Noms from cac". La fonction "Valeur actifs" renvoie toutes les informations de l'actifs, à savoir la valeur, le volume, les rendements, le poids initialisé à 0 et la moyenne des rendements entre les 2 dates. La fonction "Rendement actif" retourne la liste des tuples (date, rendement en pourcentage) d'un actif. Enfin la fonction 'RendementsPourPF' permet de créer une liste de liste contenant les rendements et valeurs de l'actif pour chaque date.

3.1.4 Classe Varcov

Dans cette classe on renvoie une matrice de variance-covariance calculée à l'aide de requêtes SQL. On récupère donc tous les rendements entre 2 dates pour tous les actifs (en parcourant les noms des actifs). On place ces rendements dans une matrice en fonction des actifs et des dates. On utilise la fonction 'np.cov' pour créer cette matrice.

3.1.5 Classe Portefeuille

La classe 'Portefeuille' est une fonction qui associe à une liste d'actifs, une valeur, une volatilité, un rendement et le score du portefeuille. La fonction 'plus petit prix' permet de renvoyer l'actif qui possède le plus petit prix parmi la liste en faisant une boucle 'for'.

La fonction 'Creation Portefeuille' permet de créer une liste d'actifs sans doublons grâce à la fonction "copy.deecopy()". Cette fonction prend également en argument l'investissement maximal à investir. Tant que la valeur maximale a investir est supérieur au prix de l'actif le moins cher et tant que la liste des indices des actifs n'est pas nulle, alors on ajoute une action au portefeuille. Pour cela on effectue une sélection aléatoire de l'actif à l'aide des index on lui attribue le nombre maximal de shares que l'on peut acheter pour cet actif et on retire l'index de la liste pour ne pas retomber dessus et on réduit l'investissement maximal suivant la part de l'actif choisi.

La fonction 'Valeur Portefeuille' permet de calculer le score associé au portefeuille grâce au portefeuille pris en argument. On utilise la valeur de chaque actif ainsi que le nombre de shares pris en compte dans le portefeuille.

La fonction ≪Poids dans portefeuille≫ définit le poids que représente l'action dans le portefeuille en récupérant la valeur de l'action multipliée par le nombre de shares que l'on divise par la valeur totale du portefeuille.

On renvoie ensuite la volatilité à l'aide de la matrice variance covariance calculée dans la classse 'VarCov'. On utilise une requête SQL et on calcule la volatilité. On récupère la matrice de variance covariance que l'on nomme 'matrice' ainsi que la matrice des poids appelée 'Listepoids'. Enfin on fait un calcul matrice de la transposée de la matrice de poids multipliée par la matrice de variance covariance multipliée par 'Listepoids'.

La fonction 'RendementsPF' permet de renvoyer le rendement global du portefeuille. On parcourt la liste des rendement et valeur créée pour chaque actif. Pour calculer le rendement du portefeuille on ajoute chaque valeur d'actif multiplié par le nombre de share et le rendement de l'actif. On fait

ensuite le produit de (1 + rendement du portefeuille à la date i de la boucle for) et on retranche 1 à la fin pour obtenir le rendement final du portefeuille.

On calcule ensuite le ratio de Sharpe en divisant le rendement par la volatilité pour estimer la qualité du portefeuille.

Enfin la fonction 'Mutation' permet la création d'un nouveau portefeuille en faisant une mutation. Pour cela on récupère la liste des actifs et on retire au maximum investi la valeur de l'actif par le nombre de shares et on réalise la même manipulation que pour la création de portefeuille. On créé un nouveau nombre de shares avec des actions random pour créer un nouveau portefeuille. Finalement, on utilise les fonctions pour associés les atttributs à ce nouveau portefeuille à savoir la volatilité, le rendement, le ratio de sharpe, les poids et la valeur.

3.1.6 Classe Population

La classe population permet de créer une liste de portefeuilles à laquelle on appliquera la fonction fitness dans la classe suivante. On utilise une fonction 'Creation Population' qui créé une population de portefeuille selon un investissement maximal, un nombre de portefeuille et une liste d'actifs. Suivant la condition de l'investissement maximal on utilise la fonction 'Creation Portefeuille' pour créer des portefeuilles aléatoires.

On créé ensuite une fonction crossover qui permet de faire les croisements et les mutations. On instancie donc deux listes de portefeuilles et on fait un croisement de ces 2 portefeuilles.

La fonction 'Crossover' prend en paramètres deux portefeuilles. Les listes d'actifs de ces deux portefeuilles sont coupées en deux et recombiné afin de crée deux nouvelles listes d'actifs. Deux nouveaux portefeuilles sont générés à partir de ces deux nouvelles listes

3.1.7 Classe AlgoG

Cette classe compose le corps de l'algorithme. On impose les attributs 'pop' et 'generation max' qui sont la population traitée et le nombre de générations générées.

La première fonction est 'sort population' qui permet de trier la population suivant le score qui lui est attribué. Pour cela, on parcourt la liste de portefeuilles et pour chaque portefeuille on tese si son score est plus petit que le suivant auquel cas on échange de place les deux portefeuilles et on continue les boucles.

La fonction 'nouvelle population' permet de créer une nouvelle population en commençant par ajouter le meilleur portefeuille, trouvé grâce à la fonction de tri, dans la nouvelle population. On utilise ensuite les fonctions de mutation et de croisement pour créer de nouveaux portefeuilles. Pour cela, on génère un random qui vaut 1 ou 2 et qui associe donc un portefeuille muté ou un portefeuille croisé. On renvoie ensuite une nouvelle liste de portefeuilles.

On y inclut une fonction algorithme génétique qui permet d'exécuter l'algorithme génétique en fonction des conditions d'objectifs de rendement et de volatilité imposés par l'utilisateur. On différencie les cas si l'utilisateur rentre un risque et une volatilité ou non.

Enfin, on effectue l'algorithme sur plusieurs générations pour terminer avec le meilleur portefeuille possible donc on instancie une fonction génération qui permet de créer une nouvelle population pour chaque génération en fonction du nombre souhaité.

3.1.8 Classe Fitness

La fonction fitness enfin calcule le fitness associé à chaque portefeuille. Ici nous avons choisi d'utiliser le ratio de Sharpe qui utilise uniquement le rendement et la volatilité et en fait le rapport.

3.1.9 Classe Main

Enfin dans le 'main' on initie la connexion avec la base de données puis on impose deux dates pour calculer les rendements sur une période donnée. On demande à l'utilisateur le montant, le rendement et la volatilité.

On récupère donc ces valeurs et on créé le nombre de portefeuilles souhaités ainsi que le nombre de générations sur lequel on teste notre algorithme. On créé nos différents actifs auxquels on associe une valeur. La population initiale est créée à l'aide de ces actifs puis on lance l'algorithme génétique qui tourne donc sur n générations et renvoie à chaque tour de boucle le portefeuille avec les actifs, leur rendement et volatilité associés ainsi que le rendement et la volatilité du portefeuille. A le nième génération on obtient le portefeuille optimisé.

3.1.10 Classe Interface

Pour créer cette interface on utilise tkinter. On créé une classe qui prend en entrées les conditions souhaitées par l'utilisateur, à savoir le maximum investi, le rendement souhaité et la volatilité maximale souhaitée. On récupère ces données à l'aide d'un '.get' qui permet à l'utilisateur de rentrer dans des cases ce qu'il souhaite.

On créé donc une fenêtre appelée 'Données' qui s'affichera sur l'écran suivant les normes que l'on lui donne (bg="grey", geometry('500x200')...). On créé ensuite un bouton 'Valider' sur lequel l'utilisateur est obligé d'appuyer avant de calculer le résultat qui n'est pas associé à une commande. Enfin, le boutton 'Calculer' va lancer le 'main' lorsque l'utilisateur cliquera dessus puisque l'on utilise 'button clicked'. On définit donc la fonction 'button clicked' comme le 'main' que nous avons expliqué précédemment et que nous avons récupéré. Une fois le 'main' exécuté, il renverra une fenêtre qui montrera les informations du portefeuille final qui s'afficheront grâce à 'showinfo'.

3.2 Modifications

1ère modification

Lors des premiers essais nous avons remarqué que les boucles via Python n'étaient pas le plus adapté pour calculer les rendements. Notre solution a été d'intégrer les rendements dans la base de données pour pouvoir les utiliser plus facilement.

Dans un premier temps nous avons décidé d'essayer de les intégrer via python. Nous avons d'abord essayé de les calculer dans la classe 'Connexion' mais nous n'avons pas réussi donc il nous a semblé plus simple d'essayer directement dans Excel.

Dans Excel nous sommes parvenus à les calculer grâce aux valeurs téléchargées dans Bloomberg. Et nous avons téléchargé le nouveau .csv dans SQL en créant une base de données.

2ème modification

Le deuxième problème rencontré concernait les portefeuilles générés. En effet, lors de la génération de nouveaux portefeuilles, ceux-là ne changeaient pas. De plus, le temps d'exécution a été réduit considérablement grâce aux modifications.

Lors de la création d'un nouveau portefeuille, l'algorithme gardait la liste d'actifs contenu dans le portefeuille précédent ainsi que le nombre d'actions (nb shares) associé. Ceci était un problème puisque lors de la création d'une population, tous les portefeuilles qu'elle contenait étaient identiques. Nous n'arrivions donc pas à générer de nouveaux portefeuilles.

Afin de remédier à ce problème de doublons nous avons importé la fonction deepcopy depuis la librairie copy.

Cette fonction nous donne la possibilité de copier les éléments du portefeuille précédent dans une autre variable. Ainsi la liste d'actifs de l'ancien portefeuille n'était pas affecté d'office à la liste en la mettant dans une variable temporaire lors de la création du nouveau portefeuille.

On utilise la fonction deepcopy() et non pas la fonction copy(), qui fonctionne seulement sur des variables dont le type existe déjà dans Python. En effet, les variables que nous utilisons sont des Actifs qui est un type de variable que nous avons créé.

Une fois la liste d'actifs copiée dans la variable, l'algorithme remet à 0 le nombre d'action de chaque action de la nouvelle liste pour générer de nouveaux nombres de shares par la suite.

3ème modification

Nous avons eu un autre problème concernant les rendements lors des essais. En effet, les rendements nous paraissaient très faible (souvent proche de 0 à 10E-5 près).

Pour cela nous avons changé la façon de calculer les rendements. Nous créons une matrice qui récupère les rendements et valeurs dans un tableau pour les utiliser dans le calcul du rendement de portefeuilles. On ajoute les rendements et valeurs à chaque date entre les deux dates sélectionnées. On obtient donc une liste de listes avec les rendements de toutes les dates pour chaque actif.

Dans la classe portefeuille, pour toutes les dates on parcourt la liste des actifs et on calcule la somme des rendements. On fait la somme des rendements des actifs, par leur valeur, par le nombre de shares le tout divisé par la valeur du portefeuille. Pour calculer le rendement du portefeuille on fait donc le produit de ces sommes + 1 et on retire 1 après les boucles for.

4ème modification

Lors de la mutation des portefeuilles nous avions un problème de calcul de ratio de Sharpe. Nous avons commencé par reprendre notre calcul du ratio de Sharpe. Le calcul du ratio de Sharpe se calcule en faisant le quotient des rendements sur le risque soit la volatilité. Toutefois même en reprenant le calcul du rendement nous n'avons pas trouvé le problème qui finalement venait de la fonction mutation.

La solution a été de reprendre totalement la fonction mutation.

4 Recettes

4.1 Résultats

Pour afficher nos résultats nous utilisons une interface graphique. On affiche une fenêtre user form dans laquelle on demande à l'utilisateur les contraintes qu'il souhaite imposer. On renvoie les valeurs du portefeuille optimisé avec la valeur générale du portefeuille, la volatilité, le rendement et le score du portefeuille calculé à l'aide du ratio de Sharpe après n générations.

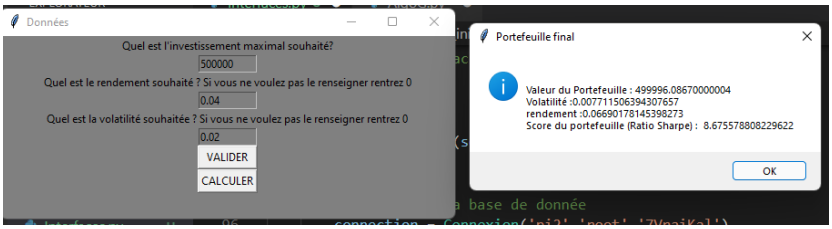


Figure 4 : Interface graphique

Avant de passer par cette interface graphique nous avons d'abord récupéré les résultats directement via la console qui nous renvoyait toutes les étapes lors des générations comme ci-dessous.

```
############## Generation : 0 ##############


CROSSOVER

Valeur du croisement sur liste 1: 488241.2081

Valeur du croisement sur liste 2 488218.72800000006


 [
Valeur du Portefeuille : 499992.7414000001
Volatilit? :0.004448531810527952
rendement :0.026660109513133533
Score du portefeuille (Ratio Sharpe) :  5.993013121776353
,
Valeur du Portefeuille : 499997.2204
Volatilit? :0.00864286040968941
rendement :0.044820814797727926
Score du portefeuille (Ratio Sharpe) :  5.185877437923194
,
Valeur du Portefeuille : 500014.9201
Volatilit? :0.009304956632146545
rendement :0.03957835854171925
Score du portefeuille (Ratio Sharpe) :  4.253470500333647
,
Valeur du Portefeuille : 499971.722
Volatilit? :0.005342670968211133
rendement :0.020445071642890333
Score du portefeuille (Ratio Sharpe) :  3.8267510323091978
,
Valeur du Portefeuille : 499997.3131
Volatilit? :0.01423483696845269
rendement :0.02883066359001183
Score du portefeuille (Ratio Sharpe) :  2.0253595916768474
]
```

```
############## Generation : 1 ##############


CROSSOVER

Valeur du croisement sur liste 1: 34858.12740000001

Valeur du croisement sur liste 2 34858.09889999999


 [
Valeur du Portefeuille : 499992.7414000001
Volatilit? :0.004448531810527952
rendement :0.026660109513133533
Score du portefeuille (Ratio Sharpe) :  5.993013121776353
,
Valeur du Portefeuille : 499997.1919
Volatilit? :0.008439291459737989
rendement :0.045305015712045524
Score du portefeuille (Ratio Sharpe) :  5.368343530755613
,
Valeur du Portefeuille : 500014.94859999995
Volatilit? :0.009027471403328163
rendement :0.03929336622276549
Score du portefeuille (Ratio Sharpe) :  4.352643665897316
,
Valeur du Portefeuille : 499992.676
Volatilit? :0.004704319258750047
rendement :0.015758607689918103
Score du portefeuille (Ratio Sharpe) :  3.3498168009339606
,
Valeur du Portefeuille : 499996.507
Volatilit? :0.01310732305481562
rendement :0.03139665303803083
Score du portefeuille (Ratio Sharpe) :  2.395352041505968
]
```

Figure 5 : Console lors de l'algorithme génétique

Finalement, la console renvoie de la même façon que pour le userform, les valeurs du portefeuille.

4.2 Code

La connexion avec la base de données s'est faite via MySQL et une classe connexion sur Python.

```sql
drop table cac;

create table cac
(
date date,
value float,
volume int,
name varchar(6),
Rendements float);

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/datacac.csv'
INTO TABLE cac
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\r\n'
Ignore 1 lines
(@date,value,volume,name,Rendements)
SET date = STR_TO_DATE(@date, '%d/%m/%Y');


select * from cac;
```

```python
import pymysql.cursors

#Classe de connexion à la BDD MySQLWorkbench
class Connexion():

    def __init__(self, database, user, mdp):
        self.database = database
        self.user = user
        self.mdp = mdp
        self.conn = None
        self.cur = None



    def initialisation(self):
        #La fonction ouvre la connexion avec la base de donnée
        self.conn = pymysql.connect(host='localhost',
                                    user=self.user,
                                    password=self.mdp,
                                    db=self.database,
                                    cursorclass=pymysql.cursors.DictCursor)
        self.cur = self.conn.cursor()



    def execute(self,requete):
        #La fonction prend en argument une requete SQL sous forme de chaine de caractère
        #Et retourne le curseur qui permet d'acceder au résultat de la requete
        self.cur.execute(requete)
        return self.cur



    def close_connection(self):
        #La fonction ferme la connexion avec la Base de donnés
        self.cur.close()
        self.conn.close()
```

Figure 6 : MySQL et connexion à la base de données

```python
#Classe d'actifs
class Actifs():

    def __init__(self, nom, valeur, volume, date,nb_shares,rendement,poids,ListeRendementsValeurs):
        self.nom = nom
        self.valeur = valeur
        self.volume = volume
        self.date = date
        self.nb_shares = nb_shares
        self.rendement = rendement
        self.poids=poids
        self.ListeRendementsValeurs=ListeRendementsValeurs



    def creationActifs(connexion):
        #Fonction qui prend en argument la connexion avec la base de données
        #La fonction retourne une liste d'actifs avec le nom de chaque actif
        #Tout les autres attributs sont initialisée à 0
        requete = 'Select distinct Noms from cac'
        curseur = connexion.execute(requete)
        liste_Actifs = []

        for row in curseur:
            action = Actifs(row['Noms'],0,0,0,0,0,0,0)
            liste_Actifs.append(action)

        return liste_Actifs
```

```
def Valeur_Actifs(self, date1, date2, connection):
    #Fonction  qui prend en arguments une liste d'actifs, la date du jour (date1),
    #la date jusqu'à laquelle on souhaite etudier la performance de l'actif (date2)
    #et la connexion avec la base de données

    #La fonction retourne une liste d'actifs, chacun asocié à une liste contenant
    #toutes les informations le concernant
    requete = "Select Valeurs,Volumes,Rendements from cac where Dates = '"+date1+"' and Noms = '"+str(self.nom)+"';"
    curseur = connection.execute(requete)
    row = curseur.fetchone()

    self.valeur = row['Valeurs']
    self.volume = row['Volumes']
    self.date = date1
    self.rendement=row['Rendements']
    self.poids=0
    self.RendementsPourPF(date1,date2,connection)

    return self



#Creation d'une liste de (rendements actif,valeur actif) par jour pour un actif sur la période donnée (date1 à date2)
def RendementsPourPF(self,date1,date2,connection):
    Liste=[]
    requete2="select Rendements,Valeurs from cac where Noms='"+str(self.nom)+"' and Dates between '"+date1+"' and '"+date2+"';"
    #on récupère la liste des Rendements de chaque actif
    curseur2=connection.execute(requete2)
    for row in curseur2:
        Liste.append([row['Rendements'],row['Valeurs']])

    #On ajoute la liste (rendements,valeur) aux parametres de l'actif
    self.ListeRendementsValeurs=Liste
```

```
#Fonction qui retourne les informations concernant une action sous forme de chaine de caratères
def __repr__(self):
    return "Nom : {0}, Valeur : {1}, Nbr d'Actions : {2}, \nDate : {3}\n".format(self.nom,self.valeur, self.nb_shares, self.date)

################################  FONCTION PAS UTILISEE ##########################################

def MoyenneRendements(self,date1,date2,connection):
    Liste=[]
    requete1="Select distinct Noms from cac;"
    curseur=connection.execute(requete1)
    ListeNoms=[]
    #on récupère la liste des noms des actifs
    for row in curseur:
        ListeNoms.append(row['Noms'])
    for name in ListeNoms:
        somme=0
        requete2="select Rendements from cac where Noms='"+name+"' and Dates between '"+date1+"' and '"+date2+"';"
        #on récupère la liste des Rendements de chaque actif
        curseur2=connection.execute(requete2)
        nbrow=0
    for row in curseur2:
        somme+=(row['Rendements'])
        nbrow+=1
    moyenne=round(somme,6)
    if(self.nom==name):
        self.moyenneRendements=moyenne

def CAGR(self, date1, date2, connexion):
    requete1 = "Select Valeurs from cac where Noms = '"+self.nom+"';"
    requete2 = "Select Valeurs from cac where Noms = '" + self.nom + "' and Dates = '"+ date1 +"';"  # Récupération de P_initial
    requete3 = "Select Valeurs from cac where Noms ='" + self.nom + "' and Dates = '"+ date2 +"';" # Récupération de P_final
    requete4 = "Select count(Valeurs) from cac where Noms = '" + self.nom + "' and Dates between '" + date1 +"' and '"+date2+"';"
    P_init = connexion.execute(requete2)
    P_final = connexion.execute(requete3)
    nb_jours = connexion.execute(requete4)
    cagr = ((P_final / P_init) **(1/nb_jours))
    return cagr
```

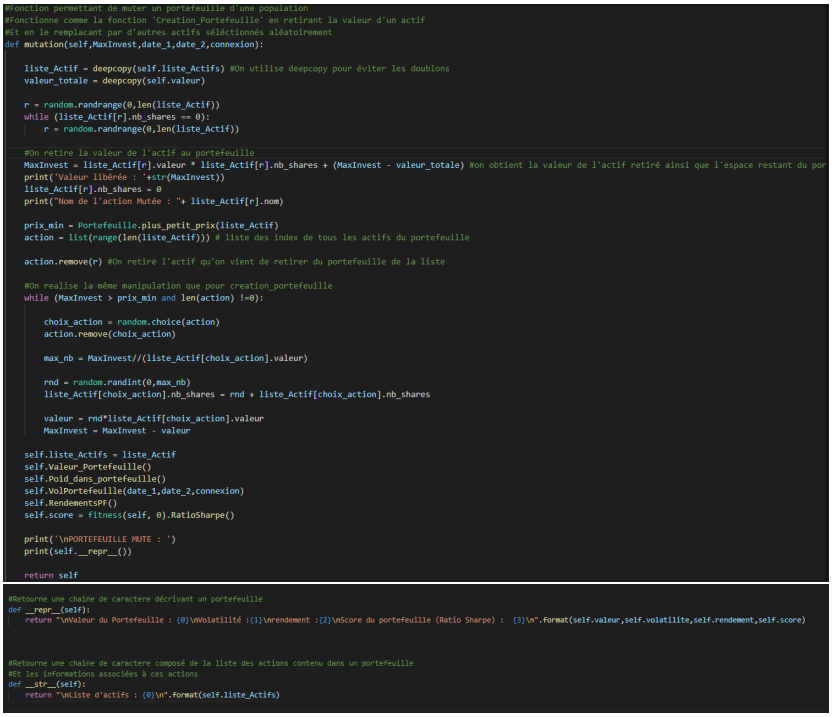
Figure 7 : Classe Actifs

```
import random
from VaRCov import VaRCov
from Fitness import fitness
import math
import numpy as np

#pour copier la liste d'actif et pas faire de doublons
from copy import deepcopy

#Classe Portefeuille
class Portefeuille():

    def __init__(self, liste_Actifs, valeur, volatilite, rendement, score):
        self.liste_Actifs = liste_Actifs
        self.valeur = valeur
        self.volatilite = volatilite
        self.rendement = rendement
        self.score = score

    # Calcul du prix de l'actif avec le moins de valeur
    def plus_petit_prix(liste_Actif):
        min = liste_Actif[0].valeur
        for asset in liste_Actif:
            if asset.valeur < min:
                min = asset.valeur
        #retourne le prix le plus faible
        return min
```

```python
#Créé un portefeuil composé d'une liste d'action aléatoire
def Creation_Portefeuille(self, MaxInvesti,date_1,date_2,connexion):

    #pour copier la liste d'actif et pas faire de doublons
    liste_Actif = deepcopy(self.liste_Actifs)
    prix_min = Portefeuille.plus_petit_prix(liste_Actif)
    action = list(range(len(liste_Actif))) #liste des index de tous les actifs du portefeuille
    #On reinitialise la valeur des nb_shares par précaution
    for i in range(len(liste_Actif)):
        liste_Actif[i].nb_shares = 0

    while (MaxInvesti > prix_min and len(action) !=0 ):
    # Tant que la valeur a investir (MaxInvest) est superieur au prix de l'actif le moins cher
    # Et tant que la liste des index des actifs du portefeuil n'est pas vide
    # on ajoute une action au portefeuille

        # Selection aléatoire d'une action via son index dans la liste d'actif
        choix_action = random.choice(action)
        action.remove(choix_action) # On retire l'index de la liste pour ne pas tomber deux fois sur le même actif

        # Nombre maximal de shares que l'on peut acheter pour cet actif
        max_nb = MaxInvesti//(liste_Actif[choix_action].valeur)
        # Selection du nombre de shares entre 0 et nb_max
        rnd = random.randint(0,max_nb)
        liste_Actif[choix_action].nb_shares = rnd

        # On reduit la valeur a investir en lui retirant la valeur des parts de l'actif choisi
        MaxInvesti = MaxInvesti - liste_Actif[choix_action].nb_shares*liste_Actif[choix_action].valeur

    self.liste_Actifs = liste_Actif
    #On associe une valeur, une volatilité ainsi qu'un rendement au portefeuille
    #Et on associe un poid a chaque actif dans le portefeuille
    self.Valeur_Portefeuille()
    self.Poid_dans_portefeuille()
    self.VolPortefeuille(date_1,date_2,connexion)
    self.RendementsPF()
    #On calcul le score du portefeuille grace a la fitness
    self.score = fitness(self, 0).RatioSharpe()
    return self
```

```python
    #Calcul la valeur d'un portefeuille
    def Valeur_Portefeuille(self):
        #Fonction qui prend en argument un portefeuille et qui calcule
        #la valeur associée à ce portefeuille
        self.valeur = 0
        for i in range(len( self.liste_Actifs)):
            #valeur = valeur_asset * nb_shares
            self.valeur = self.valeur +  self.liste_Actifs[i].valeur*int( self.liste_Actifs[i].nb_shares)
        return self

    def Poid_dans_portefeuille(self):
        #Defini le poid qu'on les actions dans le portefeuille
        for i in range(len(self.liste_Actifs)):
            self.liste_Actifs[i].poids = 0
            poids = self.liste_Actifs[i].valeur * self.liste_Actifs[i].nb_shares
            self.liste_Actifs[i].poids  = round(poids / self.valeur,5)
        return self

    #Calcule la volatilité d'un portefeuille entre deux dates
    def VolPortefeuille(self,date_1,date_2,connexion):
        self.volatilite = 0
        Listepoids=[]
        for actif in self.liste_Actifs:
            Listepoids.append(actif.poids)

        Listepoids=np.array(Listepoids)
        mat = VaRCov([])
        mat.CalculMatrice(connexion,date_1,date_2)
        matrice = mat.matrice
        vol = math.sqrt((np.transpose(Listepoids)@matrice@Listepoids))

        self.volatilite=vol
```

```python
    #Calcule le rendement d'un portefeuille (Rendements PorteFeuille)
    def RendementsPF(self):
        liste_Actif = self.liste_Actifs
        RendementPF = 0
        Liste = []
        SommePF = 1
        for j in range(0,len(liste_Actif[1].ListeRendementsValeurs) -1):
            RendementPF =0
            for i in liste_Actif:
                Liste=i.ListeRendementsValeurs
                RendementPF+=Liste[j][0]*i.nb_shares*Liste[j][1]/self.valeur
            SommePF*=(1+RendementPF)

        self.rendement =(SommePF-1)/10
```



```
#Fonction permettant de muter un portefeuille d'une population
#Fonctionne comme la fonction 'Creation_Portefeuille' en retirant la valeur d'un actif
#Et en le remplacant par d'autres actifs séléctionnés aléatoirement
def mutation(self,MaxInvest,date_1,date_2,connexion):

    liste_Actif = deepcopy(self.liste_Actifs) #On utilise deepcopy pour éviter les doublons
    valeur_totale = deepcopy(self.valeur)

    r = random.randrange(0,len(liste_Actif))
    while (liste_Actif[r].nb_shares == 0):
        r = random.randrange(0,len(liste_Actif))

    #On retire la valeur de l'actif au portefeuille
    MaxInvest = liste_Actif[r].valeur * liste_Actif[r].nb_shares + (MaxInvest - valeur_totale) #on obtient la valeur de l'actif retiré ainsi que l'espace restant du por
    print('Valeur libérée : '+str(MaxInvest))
    liste_Actif[r].nb_shares = 0
    print("Nom de l'action Mutée : "+ liste_Actif[r].nom)

    prix_min = Portefeuille.plus_petit_prix(liste_Actif)
    action = list(range(len(liste_Actif))) # liste des index de tous les actifs du portefeuille

    action.remove(r) #On retire l'actif qu'on vient de retirer du portefeuille de la liste

    #On realise la même manipulation que pour creation_portefeuille
    while (MaxInvest > prix_min and len(action) !=0):

        choix_action = random.choice(action)
        action.remove(choix_action)

        max_nb = MaxInvest//(liste_Actif[choix_action].valeur)

        rnd = random.randint(0,max_nb)
        liste_Actif[choix_action].nb_shares = rnd + liste_Actif[choix_action].nb_shares

        valeur = rnd*liste_Actif[choix_action].valeur
        MaxInvest = MaxInvest - valeur

    self.liste_Actifs = liste_Actif
    self.Valeur_Portefeuille()
    self.Poid_dans_portefeuille()
    self.VolPortefeuille(date_1,date_2,connexion)
    self.RendementsPF()
    self.score = fitness(self, 0).RatioSharpe()

    print('\nPORTEFEUILLE MUTE : ')
    print(self.__repr__())

    return self
```

```
#Retourne une chaine de caractere décrivant un portefeuille
def __repr__(self):
    return "\nValeur du Portefeuille : {0}\nVolatilité :{1}\nrendement :{2}\nScore du portefeuille (Ratio Sharpe) :  {3}\n".format(self.valeur,self.volatilite,self.rendement,self.score)



#Retourne une chaine de caractere composé de la liste des actions contenu dans un portefeuille
#Et les informations associées à ces actions
def __str__(self):
    return "\nListe d'actifs : {0}\n".format(self.liste_Actifs)
```

Figure 8 : Classe Portefeuille

```
import numpy as np

class VaRCov():
    def __init__(self, matrice):
        self.matrice=matrice


    #Calcul de la matrice de variance covariance
    def CalculMatrice(self,Connexion,date1,date2):
        Liste=[]
        requete1="Select distinct Noms from cac;"
        curseur=Connexion.execute(requete1)
        ListeNoms=[]
        #on récupère la liste des noms des actifs
        for row in curseur:
            ListeNoms.append(row['Noms'])

        for noms in ListeNoms:
            ListeRendements=[]
            requete2="select Rendements from cac where Noms='"+noms+"' and Dates between '"+date1+"' and '"+date2+"';"
            #on récupère la liste des Rendements de chaque actif
            curseur2 = Connexion.execute(requete2)
            for row in curseur2:
                ListeRendements.append(row['Rendements'])
            if len(ListeRendements) != 7:
                Liste.append(np.array(ListeRendements[0:7]))
            else : Liste.append(np.array(ListeRendements))

        Liste = np.array(Liste)
        matrix=np.cov(Liste,bias=True)
        self.matrice=matrix
```

Figure 9 : Classe VarCov

```
from copy import deepcopy
from Portefeuille import Portefeuille
from fitness import fitness

#Classe Population
class Population() :

    def __init__(self, list_portefeuille):
        self.list_portefeuille = list_portefeuille

    def creation_population(self, list_asset, MaxInvest, nb_portefeuille,date_1,date_2,connexion):
    #Creation de la population en fonction du nombre de portefeuille voulu.
        list_portefeuille = []
        for i in range(nb_portefeuille):
            #Creation de chaque portefeuille de la population
            p = Portefeuille(list_asset,0,0,0,0).Creation_Portefeuille(MaxInvest,date_1,date_2,connexion)
            list_portefeuille.append(p) #On ajoute le portefeuille à la liste de portefeuille qui va composer la population

        self.list_portefeuille = list_portefeuille
        return self

    #Retourne une chaine de caractère contenant la liste de portefeuilles contenu dans une population.
    def __repr__(self):
        return "{0}".format(self.list_portefeuille)
```

```python
def crossover(portefeuille_1, portefeuille_2,date_1,date_2,connexion):

    list_1 = deepcopy(portefeuille_1.liste_Actifs)
    list_2 = deepcopy(portefeuille_2.liste_Actifs)
    shares_1 = []

    for i in range(len(list_1)):
        if(list_1[i].nb_shares != 0):
            shares_1.append(list_1[i])

    shares_1 = shares_1[0:len(shares_1)//2]
    value_1 = 0

    for actif in shares_1:
        value_1 += actif.nb_shares*actif.valeur
    print('Valeur du croisement sur liste 1: '+str(value_1))

    shares_2 = []
    v2 = 0

    for i in range(len(list_2)):
        if(list_2[i].nb_shares != 0):
            shares_2.append(list_2[i])

    shares_2 = shares_2[::-1]
    mut2 = []

    for j in range(len(shares_2)):
        nb = 0
        while ((shares_2[j].valeur * nb) < value_1) and nb <= shares_2[j].nb_shares:
            nb += 1
        actif = deepcopy(shares_2[j])
        actif.nb_shares = nb-1
        value_1 -= actif.valeur * actif.nb_shares
        v2 += actif.nb_shares * actif.valeur
        mut2.append(actif)

    print('\nValeur du croisement sur liste 2',v2,'\n')

    mut1 = deepcopy(shares_1)

    for i in range(len(list_1)):
        for j in range(len(shares_1)):
            if list_1[i].nom == shares_1[j].nom:
                list_1[i].nb_shares -= shares_1[j].nb_shares
```

```python
    print('\nValeur du croisement sur liste 2',v2,'\n')

    mut1 = deepcopy(shares_1)

    for i in range(len(list_1)):
        for j in range(len(shares_1)):
            if list_1[i].nom == shares_1[j].nom:
                list_1[i].nb_shares -= shares_1[j].nb_shares

    for i in range(len(list_2)):
        for j in range(len(mut2)):
            if list_2[i].nom == mut2[j].nom:
                list_2[i].nb_shares -= mut2[j].nb_shares

    for i in range(len(list_1)):
        for j in range(len(mut2)):
            if list_1[i].nom == mut2[j].nom:
                list_1[i].nb_shares += mut2[j].nb_shares

    for i in range(len(list_2)):
        for j in range(len(mut1)):
            if list_2[i].nom == mut1[j].nom:
                list_2[i].nb_shares += mut1[j].nb_shares

    final_mut1 = Portefeuille(list_1,0,0,0,0)
    final_mut2 = Portefeuille(list_2,0,0,0,0)


    final_mut1.Valeur_Portefeuille()
    final_mut1.Poid_dans_portefeuille()
    final_mut1.VolPortefeuille(date_1,date_2,connexion)
    final_mut1.RendementsPF()
    #On calcul le score du portefeuille grace a la fitness
    final_mut1.score = fitness(final_mut1, 0).RatioSharpe()

    final_mut2.Valeur_Portefeuille()
    final_mut2.Poid_dans_portefeuille()
    final_mut2.VolPortefeuille(date_1,date_2,connexion)
    final_mut2.RendementsPF()
    #On calcul le score du portefeuille grace a la fitness
    final_mut2.score = fitness(final_mut2, 0).RatioSharpe()

    return [final_mut1,final_mut2]
```

Figure 10 : Classe Population

```python
from Portefeuille import Portefeuille
from Population import Population
import random
from copy import deepcopy

class AlgoG() :

    def __init__(self, pop, generation_max):
        self.pop = pop
        self.generation_max = generation_max

    #On trie la population en fonction de son score en ordre descendant
    def sort_population(self):
        list_score=[]
        list_portefeuille = deepcopy(self.pop.list_portefeuille)
        for i in range(len(list_portefeuille)):
            list_score.append(list_portefeuille[i].score)
        list_score = sorted(list_score,reverse=True)

        final_list = []
        for i in range(len(list_score)):
            for j in range(len(list_portefeuille)):
                if(list_portefeuille[j].score == list_score[i]):
                    final_list.append(list_portefeuille[j])
        self.pop.list_portefeuille = final_list
        return self
```

```python
    def nouvelle_population(self,list_asset,MaxInvest,date_1,date_2,connexion):
    #Creation d'une nouvelle population en fonction d'une population précédente
        pop_precedente = self.pop

        Pourcentage_garder = 0.5 #Pourcentage de la population précédente à muter et croiser
        new_list_portefeuille = []
        new_list_portefeuille.append(pop_precedente.list_portefeuille[0]) #On garde le portefeuille avec le meilleur score.

        i = 1
        while i < len(pop_precedente.list_portefeuille):

            if i < 2:#len(pop_precedente.list_portefeuille)*Pourcentage_garder:
                #On croise et mute le pourcentage de la population choisit

                rnd = round(random.uniform(1,2)) #Une chance sur deux de croiser et une chance sur deux de muter
                if rnd == 1:
                    print('\nCROSSOVER\n')
                    cross = Population.crossover(pop_precedente.list_portefeuille[i],pop_precedente.list_portefeuille[i+1],date_1,date_2,connexion)
                    new_list_portefeuille.append(cross[0])
                    new_list_portefeuille.append(cross[1])
                    i+=2
                if rnd == 2:
                    print('\nMUTATION\n')
                    new_list_portefeuille.append(pop_precedente.list_portefeuille[i].mutation(MaxInvest,date_1,date_2,connexion))
                    i+=1
            else :
                #On complete la population avec des portefeuilles crées aléatoirement
                new_list_portefeuille.append(Portefeuille(list_asset,0,0,0,0).Creation_Portefeuille(MaxInvest,date_1,date_2,connexion))
                i +=1

        return new_list_portefeuille
```

```python
#fonction executant l'algorithme genetique en fonction des conditions d'objectifs de rendement et de volatilité posés par l'utilisateur
def algorithme_genetique(self, liste_assets, max_invest, exp_ret, exp_vol,date_1,date_2,connexion):
    generation = 0

    self.sort_population()

    if (exp_ret == 0):
        if (exp_vol == 0):
        #Si aucun rendement ou volatilité n'est entré par l'utilisateur.
            while generation <= self.generation_max:

                self.Generation(liste_assets, max_invest, generation, date_1,date_2,connexion)
                generation += 1

            return self.pop.list_portefeuille[0]

        #Si aucun rendement n'est entré par l'utilisateur (mais l'utilisateur à rentré une volatilité).
        else :
            while generation <= self.generation_max and (self.pop.list_portefeuille[0].volatilite < 0.9*exp_vol or self.pop.list_portefeuille[0].volatilite > 1.1*exp_vol):

                self.Generation(liste_assets, max_invest, generation, date_1,date_2,connexion)
                generation += 1

            return self.pop.list_portefeuille[0]
    else:
        if (exp_vol == 0):
        #Si aucune volatilité n'est entré par l'utilisateur (mais l'utilisateur à rentré un rendement)
            while generation <= self.generation_max and (self.pop.list_portefeuille[0].rendement < 0.9*exp_ret):

                self.Generation(liste_assets, max_invest, generation, date_1,date_2,connexion)
                generation += 1

            return self.pop.list_portefeuille[0]
        else :
        #Si un rendement et une volatilité sont entré par l'utilisateur.
            while generation <= self.generation_max and ((self.pop.list_portefeuille[0].rendement < 0.9*exp_ret) or (self.pop.list_portefeuille[0].volatilite < 0.9*exp_vol or self.pop.list_portefeuille[0].volatilite > 1.1*exp_vol)):

                self.Generation(liste_assets, max_invest, generation, date_1,date_2,connexion)
                generation += 1

            return self.pop.list_portefeuille[0]


#Crée la nouvelle population pour chaque génération
def Generation(self, liste_assets, max_invest, generation, date_1,date_2,connexion):

        print('\n############ Generation : '+str(generation)+' ############\n')
        self.pop = Population(self.nouvelle_population(liste_assets,max_invest,date_1,date_2,connexion))
        self.sort_population()

        print('\n',self.pop.__repr__())
```

Figure 11 : Classe AlgoG

Enfin, tout au long de ce projet nous avons travaillé en partageant nos avancées sur Github. Nous avons chacun créé des branches et seul le projet final a été push sur le master en fin de projet.

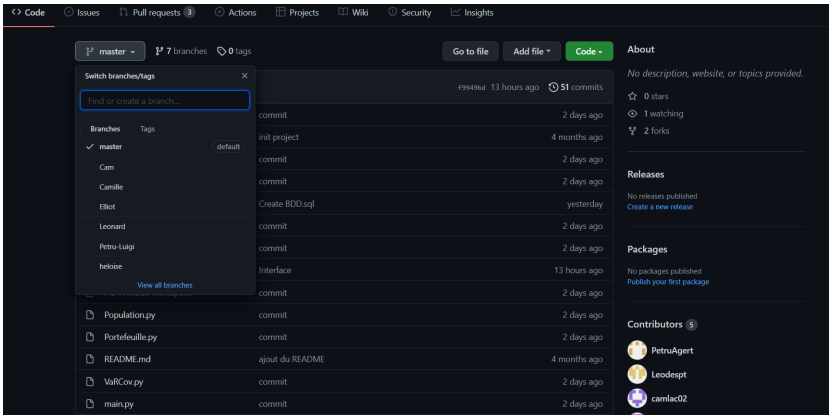


Figure 9 : Github