Département de génie informatique et de génie logiciel

INF8970

Projet final en génie informatique

Rapport de mi-session

1chipML

Équipe No 7

Félix-Antoine Constantin
Paul-André Bisson
Nicholas Legrand
Florence Cloutier
Alexis Foulon
Gaya Mehenni

24 février 2023

Table des matières

1. Propositions de solutions (Q2.2) **5**

1.1 Descriptions de quelques solutions possibles 5

1.1.1 Solution générale 5

1.1.2 Transformée de Fourier rapide (FFT) et transformée de Fourier discrète (DFT) 5

1.1.3 Lanczos 5

1.1.4 Jacobi 6

1.1.5 Descente de gradient 6

1.1.6 Algorithme génétique 6

1.1.7 Algorithme de Monte Carlo 7

1.2 Explications du rejet de certaines solutions ou de facteurs justifiant le choix de celle retenue 8

1.2.1 Solution générale 8

1.2.2 Transformée de Fourier rapide (FFT) et transformée de Fourier discrète (DFT) 8

1.2.3 Lanczos 9

1.2.4 Jacobi 9

1.2.5 Descente de gradient 9

1.2.6 Algorithme génétique 10

1.2.7 Algorithme de Monte Carlo 10

2. Description de l'architecture du système retenue (Q2.3 – Q4.2 – Q4.3) **11**

2.1 Détails de la solution retenue 11

2.1.1 Solution générale 11

2.1.2 Transformée de Fourier rapide (FFT) et transformée de Fourier discrète (DFT) 11

2.1.3 Lanczos 11

2.1.4 Jacobi 11

2.1.5 Descente de gradient 12

2.1.6 Algorithme génétique 12

2.1.7 Algorithme de Monte Carlo 13
2.2 Quelques diagrammes de classes ou de modules 13
2.3 Diagrammes d'états et/ou d'interface usager 14
2.4 Interfaces logicielles et/ou matérielles importantes 15
2.5 Simulation, norme ou modèles importants 16
2.6 Description d'outils, librairies ou cadre de travail (framework) utilisés (Q5.1) 16
2.7 Quelques méthodes de test à appliquer pour valider la solution. (Q4.4) 17
2.7.1 Transformée de Fourier rapide (FFT) et transformée de Fourier discrète (DFT) 17
2.7.2 Descente du gradient 18
2.7.3 Génétique 18
2.7.4 Lanczos 18
2.7.5 Jacobi 19
2.7.6 Monte Carlo 19
3. Gestion du projet **19**
3.1 Identification des tâches principales du projet (Q3.3) 19
3.1.1 Implémentation 20
3.1.2 Recherche 20
3.1.3 Optimisation 20
3.1.4 Prototype 21
3.1.5 Documentation 21
3.2 Répartition des tâches et responsabilités dans l'équipe 21
3.3 Mesure sommaire de la progression du projet 23
3.4 Mesures de révision du travail réalisé par les autres membres de l'équipe 25
4. Progression jusqu'à la fin du projet et conclusion **26**
4.1 Retour sommaire sur le travail déjà réalisé 26
4.2 Identifications des facteurs encore mal connus qui pourraient poser problème d'ici la fin du projet (Q2.6) 28
4.3 Principales tâches à réaliser pour assurer le succès du projet et que l'équipe estime pouvoir compléter avant la fin (Q3.1) 28
4.3.1 Implémentation 28
4.3.2 Exécution sur Arduino 29
4.3.3 Optimisation 29

4.3.4 Recherche 29
4.3.5 Prototypes 29
4.3.6 Documentation 30
4.3.7 Itérations avec le client 30
5. Références utilisées (Q3.2) **31**
6. Annexe **32**
Annexe 1 : Fichier de formatage .clang-format 32
Annexe 2 : Lien pour accéder au tag présent sur le répertoire GitHub du projet: 33

1. Propositions de solutions (Q2.2)

1.1 Descriptions de quelques solutions possibles

1.1.1 Solution générale

La solution finale de la librairie peut être rendue au client de plusieurs manières. Effectivement, une librairie statique (.a) ou dynamique (.so) peut être compilée et distribuée permettant ensuite aux programmes de s'y lier lors de la compilation. D'autre part, les fichiers sources peuvent également être rendus au client afin qu'il ait un contrôle complet sur l'implémentation des algorithmes. L'entreprise a une vision d'une librairie à code ouvert pour utilisation industrielle pour des microcontrôleurs. Les variantes possibles de cette librairie pourraient être une librairie en Python ou en C. De plus, la librairie pourrait posséder des dépendances entre les fichiers, comme avoir un fichier «utils.h» utilisé par plusieurs autres fichiers ou bien la librairie pourrait être composée seulement de fichiers indépendants entre eux.

1.1.2 Transformée de Fourier rapide (FFT) et transformée de Fourier discrète (DFT)

L'algorithme de la FFT est généralement utilisé pour transformer un signal d'un certain domaine, comme le temps, vers un domaine de fréquences.

Il existe plusieurs solutions possibles en ce qui concerne l'implémentation des algorithmes FFT et DFT. Pour la DFT et la FFT, il existe de nombreuses techniques visant à accélérer les différents calculs qui ont lieu. La première technique est de calculer à l'avance les facteurs multiplicatifs, aussi connus sous le nom de « Twiddle factors » et de les utiliser après, lors des multiplications. En se penchant plus sur la FFT, il existe de nombreuses implémentations qui pourraient répondre aux besoins du client. Tout d'abord, l'implémentation la plus répandue est d'utiliser de la récursion afin de résoudre la transformée de Fourier. Sinon, il existe des implémentations itératives, mais plus complexes à implémenter [1].

1.1.3 Lanczos

L'algorithme de Lanczos est utilisé pour calculer les vecteurs propres et valeurs propres associées à une matrice symétrique autoadjointe. Cet algorithme prend en entrée une matrice et retourne 2 matrices en sortie. Ces matrices ne contiennent pas directement les vecteurs propres et valeurs propres, mais fournissent plutôt des matrices dont ces valeurs sont plus simples à calculer. Plus exactement, l'algorithme retourne une matrice tridiagonale qui permet de plus facilement calculer les valeurs propres et une seconde matrice utilisée pour transformer les vecteurs propres vers l'espace initial.

L'algorithme est assez simple à implémenter et peu de variantes existent à son sujet. En effet, le seul défi que cet algorithme pose est l'instabilité numérique produite par son exécution. Pour pallier ce problème, de nombreuses méthodes existent. L'ordre des opérations peut notamment être modifié. C. Paige [2] a démontré que l'ordre des opérations avait un impact sur l'instabilité numérique produite par l'algorithme. Une attention particulière à donc été portée à celle-ci. De plus, de nombreuses méthodes existent pour tenter de garder la matrice de vecteurs propre orthonormée et donc

d'éviter de perdre certaines caractéristiques importantes de l'algorithme. Par exemple, une réorthogonalisation peut être effectuée sur la matrice.

1.1.4 Jacobi

L'algorithme de Jacobi est un algorithme de résolution de valeurs propres et de vecteurs propres. En effet, étant donné une matrice symétrique, l'algorithme est capable de calculer les vecteurs et valeurs propres de la matrice.

Plusieurs implémentations de cet algorithme existent, chacune avec ses propres avantages. Par exemple, la méthode de Jacobi cyclique se retrouve à être plus efficace dans le cas de certaines matrices et la méthode classique l'est dans d'autres cas. La différence principale entre ces deux méthodes réside dans le choix de l'élément à éliminer lors d'une itération de l'algorithme. Puis, des implémentations existent pour être plus efficaces en moyenne dans le temps. Cependant, ces dernières utilisent plus d'espace et sont parfois sous la protection de droits d'auteurs. Le fait d'implémenter l'algorithme sur un microcontrôleur implique une bonne gestion de la mémoire et de l'efficacité de l'algorithme. Notamment, la version présentée en Fortran dans le livre *Numerical Recipes* [1] semble trouver un juste milieu entre simplicité d'implémentation, efficacité et utilisation de la mémoire.

1.1.5 Descente de gradient

L'algorithme de descente du gradient est une technique d'optimisation permettant de minimiser une fonction quelconque. Cet algorithme commence à un point de départ initial et va tenter de converger vers un minimum à partir de ce point en se déplaçant successivement dans la direction inverse du gradient de la fonction. Lorsque la descente du gradient converge vers un minimum, il n'y a aucune garantie que celui-ci soit un minimum global. En effet, à cause de l'utilisation du gradient pour déterminer la direction du déplacement il est parfaitement possible que la descente du gradient aboutisse dans un minimum local.

Plusieurs variantes d'implémentation de cet algorithme existent. En effet on peut implémenter une descente du gradient classique qui consiste à calculer le gradient à notre point initial, se déplacer à l'inverse de cette direction vers un nouveau point, recalculer le gradient au nouveau point, et ainsi de suite. Une seconde variante très intéressante de cet algorithme est celle de la descente du gradient conjugué dont une version est présentée dans le livre *Numerical Recipes* [1]. Cette variante ajuste la direction de la descente par un conjugué afin de rendre celle-ci plus optimale.

Enfin, puisque la méthode de la descente du gradient nécessite le calcul de dérivées, une autre considération à prendre en compte est le calcul de celles-ci. Plusieurs options s'offrent à nous. La première est de demander à l'utilisateur de lui-même fournir une fonction calculant la dérivée de la fonction qu'il veut optimiser. La seconde est d'utiliser une approximation de la dérivée grâce à la méthode des différences finies. Il est possible de faire une approximation de premier ordre ou de second ordre pour cette différence selon la fonction spécifique dont on veut approximer la dérivée [5].

1.1.6 Algorithme génétique

L'algorithme génétique est une métaheuristique utilisée pour des problèmes d'optimisation globaux, c'est-à-dire qu'il permet d'optimiser des fonctions qui possèdent

plusieurs optimums locaux, ce qui est sa plus grande force face à un algorithme d'escalade traditionnel. Pour ce faire, l'algorithme simule le comportement de l'évolution d'une population. Initialement, une population est créée de manière aléatoire. Ensuite, au travers de diverses itérations, une série d'opérations génétiques est appliquée de manière à optimiser les résultats obtenus. Les opérations appliquées sur la population sont habituellement le croisement et la mutation. Le croisement est utilisé pour créer des enfants à partir de deux membres de la population qui sont choisis en fonction de leur force. La force d'une solution est évaluée à l'aide d'une fonction objective. Les nouveaux enfants ont ensuite la possibilité de subir une mutation aléatoire de l'un de leurs gènes. Lorsque suffisamment d'enfants sont créés, la génération précédente est remplacée. Le processus se répète jusqu'à ce qu'une solution acceptable soit produite ou lorsque le nombre d'itérations maximum est atteint [7].

Cet algorithme peut toutefois consommer énormément de mémoire pour l'enregistrement de la population , la génération des enfants et des forces de chaque membre de la population. Une deuxième limite de cet algorithme est qu'il peut trouver rapidement une bonne solution, mais il peut prendre beaucoup d'itérations pour trouver l'optimum global [7].

Plusieurs variantes de cet algorithme existent. Une des différences clés de ces variantes est le processus de sélection des parents. Une des méthodes de choix les plus populaires est la sélection des parents proportionnés, c'est-à-dire que les parents avec la meilleure aptitude vont davantage être sélectionnés. Une deuxième méthode de sélection qui existe est la sélection par tournoi. L'algorithme choisit aléatoirement un certain nombre de membres de la population et retourne le meilleur membre. La dernière méthode de sélection est la sélection par rang. Cette variante est légèrement différente de la méthode de sélection proportionnée. Elle classe les solutions selon leur aptitude et choisit ensuite les parents de manière aléatoire. Les chances qu'une solution soit choisie augmentent avec son rang. Une autre variante populaire de cet algorithme implémente l'élitisme, c'est-à-dire de directement passer les meilleures solutions aux prochaines générations. [7]

Le croisement des parents peut également être fait de plusieurs façons. L'utilisation d'un index aléatoire pour déterminer le point de croisement est une des méthodes les plus utilisées. Une autre méthode intéressante est un croisement uniforme, cela veut dire qu'on choisit aléatoirement chaque gène d'un des deux parents. [7]

1.1.7 Algorithme de Monte Carlo

L'algorithme de Monte Carlo appliqué à l'apprentissage par renforcement est un algorithme permettant d'apprendre quelle action future serait la plus favorable dans le contexte actuel. L'algorithme détermine cette action en simulant aléatoirement des épisodes de l'arbre d'actions et en évaluant cet épisode grâce à un système de récompenses. Un épisode est une simulation jusqu'à un nœud terminal de l'arbre. Lorsque la récompense obtenue est convenable selon un certain standard précisé par l'utilisateur, la simulation prend fin et retourne l'action menant à la récompense souhaitée.

La méthode de Monte Carlo est utile lorsque l'arbre d'actions possible est trop grand pour le parcourir en entier dans un temps raisonnable. Cette méthode permet de

simuler plusieurs parcours de l'arbre en priorisant les parcours qui semblent plus favorables selon les récompenses obtenues dans les épisodes précédents.

Plusieurs variantes de cet algorithme existent soit la variante avec départs exploratoires, qui émet l'hypothèse que chaque nœud de l'arbre commence avec une probabilité non nulle d'être sélectionné pour être le point départ de l'épisode, et celle sans départs exploratoires, qui utilise soit une stratégie avec politique absente ou une stratégie avec politique présente [8]. Ces variantes sont décrites dans le manuel d'*Introduction à l'apprentissage par renforcement* de Richard S. Sutton et Andrew G. Barto [8]. De plus, il existe plusieurs méthodes de récompenses pour entraîner l'algorithme Monte Carlo, comme la méthode UCB ou la méthode gourmande [8].

Cet algorithme a le potentiel de consommer beaucoup de mémoire puisqu'il crée un arbre d'actions et de récompenses associées. Plus il y a de simulations, plus cet arbre prend de l'expansion. Une seconde limite de cet algorithme est le facteur aléatoire. En effet, en se basant sur des épisodes aléatoires, l'algorithme peut éventuellement ne jamais parcourir des branches de l'arbre qui sont en fait les plus favorables. Ce problème est connu communément comme étant la maintenance de l'exploration, qui résulte en l'obtention de minimums locaux.

1.2 Explications du rejet de certaines solutions ou de facteurs justifiant le choix de celle retenue

1.2.1 Solution générale

Selon les spécifications du client, il est évident qu'il souhaite que le projet soit clair et facile à utiliser. En effet, il a mentionné qu'il ne veut pas que les utilisateurs aient à faire des commandes spécifiques afin de faire fonctionner le projet, ni même à installer différentes librairies ou différents modules. Pour ces raisons, les solutions de créer une librairie *.a* ou *.so* à partir de commandes ont été rejetées. La solution retenue est donc d'uniquement rendre disponibles les fichiers sources sur l'entrepôt GitHub et donc de permettre à l'utilisateur de les ajouter manuellement à son projet. Également, puisque le client désire avoir des fonctionnalités optimisées pour microcontrôleur, il a été décidé de développer la librairie en C. En effet, C est reconnu pour être plus rapide que Python et procure donc un avantage significatif par rapport à ce dernier. De plus, puisque le client ne souhaite pas que des librairies externes soient utilisées, il n'est pas possible de tirer avantage des nombreuses librairies offertes par Python. En ce qui concerne la dépendance entre les fichiers, il a été décidé de créer des fichiers utilitaires afin de réduire la duplication de code et de permettre des améliorations qui affectent plusieurs algorithmes positivement en même temps.

1.2.2 Transformée de Fourier rapide (FFT) et transformée de Fourier discrète (DFT)

Tout d'abord, la solution de calculer les facteurs multiplicatifs « Twiddle factors » à l'avance a été rejetée en raison des contraintes matérielles du client. En effet, l'utilisation de cette solution impliquerait d'utiliser de la mémoire supplémentaire. Cependant, puisque les microcontrôleurs visés, tels que les Arduinos UNO, possèdent très peu de mémoire et que les algorithmes de FFT et DFT prennent généralement un très grand nombre de données en paramètres, il a été jugé préférable de ne pas

prendre cette route afin de satisfaire les besoins du client. La même logique a été utilisée pour le choix de l'algorithme itératif à la place de l'algorithme récursif pour la FFT. Bien que ce choix complexifie la tâche, il nous permet d'être assurés que les contraintes seront satisfaites.

1.2.3 Lanczos

Comme mentionné précédemment, peu de variantes existent pour l'algorithme de Lanczos. Puisque la librairie a pour but de trouver les vecteurs et valeurs propres avec une précision assez élevée, nous avons pris la décision de rejeter l'algorithme classique et d'implémenter la variante qui permet d'obtenir une matrice finale orthonormée. Ceci implique donc un coût de calcul supplémentaire, mais permet d'obtenir une matrice avec des résultats plus précis. Nous avons aussi pris la décision de baser l'ordre des opérations sur celle proposée par C. Paige [2] puisqu'elle réduit l'instabilité numérique sans aucun surcoût. Nous n'avions donc pas de raison de sélectionner un ordre différent qui aurait donné des résultats moins fiables.

1.2.4 Jacobi

La solution retenue est grandement basée sur celle du livre *Numerical Recipes* [1]. En effet, bien que l'algorithme donné en Fortran possède plusieurs avantages, c'est une version basée sur les formules énoncées qui est implémentée dans la librairie. L'algorithme implémenté dans la librairie utilise la multiplication de matrices pour faciliter la compréhension. Cela permettra dans le futur d'améliorer l'efficacité de l'algorithme dans le cas où un algorithme plus efficace est développé pour la multiplication de matrices. Cependant, dans le cas de cette implémentation de l'algorithme, certaines fonctions sont sensibles à l'instabilité numérique de la machine. La version du cours de Jim Lambers présenté à Stanford [4], quant à elle, semble éviter ces problèmes d'instabilité numérique. De plus, par rapport aux versions cycliques ou traditionnelles, puisqu'il y a aucune d'entre elles qui sont plus efficaces en mémoire et en temps dans tous les cas, elles ne peuvent pas être rejetées complètement.

1.2.5 Descente de gradient

La solution retenue pour l'implémentation de l'algorithme de descente du gradient est celle de la descente du gradient conjuguée. En effet, contrairement à la méthode de descente du gradient traditionnelle, cette méthode converge en moins d'itérations vers la solution, ce qui permettra d'économiser des cycles de calcul sur les microcontrôleurs. En effet, cette méthode devrait apporter un gain en vitesse de calcul d'un facteur d'environ N (soit le nombre de dimensions de la fonction à optimiser) sur l'algorithme classique [1].

Pour ce qui est de la méthode de calcul de la dérivée de la fonction à optimiser afin d'obtenir le gradient de celle-ci, il a été retenu d'implémenter les trois méthodes proposées. En effet, afin de donner plus de flexibilité à l'utilisateur de la librairie, il lui sera possible de spécifier une fonction calculant la dérivée de la fonction à calculer ou de choisir d'utiliser une approximation de premier ou second ordre avec la méthode des différences finies.

1.2.6 Algorithme génétique

Il existe plusieurs variantes de l'algorithme génétique. Le premier choix qui a dû être fait est le choix de la méthode de croisement pour la génération d'enfants. Il existe des méthodes de croisement en un point ou des méthodes de croisement multipoints. La méthode de croisement retenue est celle du croisement uniforme. Chaque gène du parent est choisi aléatoirement pour un des enfants et le choix opposé est fait pour le deuxième enfant généré. Cette méthode de croisement a été retenue parce qu'elle est très efficace et peut être calculée rapidement, ce qui est important parce que le calcul s'effectue sur un microcontrôleur. De plus, ce mode de croisement augmente la diversité des enfants et permet de sortir des optimums locaux. Les solutions créées par la méthode de croisement par index ne possèdent pas cette diversité.

Un autre choix qui a été retenu est la variante de l'élitisme. Les meilleurs éléments de chaque génération sont directement copiés dans la génération future. Cette variante a grandement amélioré la qualité des solutions trouvées par l'algorithme et a donc été retenue.

La méthode de sélection des parents par rang a été rejetée parce que le tri de la population peut prendre beaucoup de temps. La méthode de sélection par proportionnalité à été rejetée parce qu'elle peut être utilisée pour optimiser des problèmes de maximisation, mais ne peut pas optimiser des problèmes de minimisation ce qui correspond à un des tests. La décision a donc été prise d'utiliser la méthode de sélection par tournoi en raison de son temps de calcul extrêmement bas et sa capacité à minimiser des fonctions.

1.2.7 Algorithme de Monte Carlo

L'algorithme de Monte Carlo possède des enjeux de mémoire et de temps. Cela dit, il faut prioriser les variantes de l'algorithme qui sont économiques en termes de mémoire et de temps. Un premier choix qui a été fait pour l'algorithme de Monte Carlo est celui de toujours recalculer la valeur d'un nœud de l'arbre. Ainsi, il n'est pas nécessaire de garder en mémoire la valeur de chaque nœud de l'arbre, ce qui peut nécessiter beaucoup d'espace. Il y a donc un compromis entre la mémoire et le temps qui est fait ici, puisqu'il y aura du temps ajouté lors du calcul répété des valeurs des nœuds. Un second choix qui a été fait lors de la conception de cet algorithme est le choix de la méthode permettant de calculer la valeur d'un nœud selon les récompenses obtenues et le nombre de visites de celui-ci. Ce choix s'est arrêté sur la méthode UCB, puisqu'elle permet de prioriser les nœuds qui n'ont pas été visités encore, ce qui diminue les chances de manquer des branches qui pourraient être avantageuses. En comparaison, la méthode gloutonne ne permet pas de prioriser ces nœuds, d'où le choix de la méthode UCB. Un autre choix qui a été fait est celui concernant les départs exploratoires. Le choix s'est arrêté sur l'implémentation de l'algorithme avec des départs exploratoires, au lieu de l'implémentation sans ceux-ci. Ce choix a été fait pour des raisons de simplicité. En accordant une probabilité non nulle à tous les nœuds d'être choisis pour l'exploration, cela évite le problème de maintenance d'exploration. Ce problème survient lorsqu'une section importante de l'arbre n'est pas visitée. En contrepartie, la méthode sans départs exploratoires est plus utile lorsque le modèle d'apprentissage apprend directement de ses interactions avec l'environnement. Dans le

cas du projet, cette méthode n'est pas nécessaire, car l'environnement est connu d'avance et son apprentissage ne dépend pas de l'interaction avec ce dernier.

2. Description de l'architecture du système retenue (Q2.3 – Q4.2 – Q4.3)

2.1 Détails de la solution retenue

2.1.1 Solution générale

La solution retenue est de rendre à disposition au client uniquement les fichiers sources des algorithmes, sans génération de librairie sous le format .a ou .so. Plus précisément, un fichier d'en-tête est chargé d'inclure tous les algorithmes développés, afin de faciliter l'inclusion de ceux-ci si nécessaire. De plus, chaque algorithme possède son propre en-tête pour permettre au client et aux futurs utilisateurs d'inclure uniquement les algorithmes dont ils ont besoin. La librairie est entièrement développée en C et n'utilise aucune librairie externe. Également, les algorithmes qui utilisent les mêmes fonctions utilitaires vont dépendre des mêmes fichiers utilitaires.

2.1.2 Transformée de Fourier rapide (FFT) et transformée de Fourier discrète (DFT)

Comme il a été mentionné précédemment, la DFT et la FFT sont implémentées sans calculer à l'avance le facteur multiplicatif utilisé pour les calculs. La DFT a tout de même besoin de vecteurs supplémentaires afin de contenir les résultats calculés avant de le retourner. La FFT, quant à elle, a besoin d'utiliser le renversement de bits [1] afin de faciliter le calcul de cette dernière. Ensuite, afin de réduire le nombre de calculs effectués par la FFT, la récurrence trigonométrique a été utilisée, permettant de réduire considérablement le nombre d'appels aux fonctions plus lentes, telles que sin et cos. Enfin, un paramètre a été ajouté à l'algorithme afin de permettre la FFT inverse, qui est souvent nécessaire pour différentes applications, comme la convolution d'image.

2.1.3 Lanczos

L'algorithme de Lanczos a été implémenté et l'ordre des opérations a été basé sur la variante présentée par C. Paige [4]. Ceci nous permet donc de minimiser l'instabilité numérique et d'ainsi obtenir des approximations qui sont les plus fiables possibles. De plus, l'algorithme de « Gram-Schmidt » a aussi été ajouté pour permettre de réduire une fois de plus l'instabilité numérique des opérations. Puisque le surcoût associé à ce processus est relativement faible, cet algorithme est exécuté après chaque itération pour réorthogonaliser notre matrice de vecteur et donc s'assurer de maximiser la précision des résultats.

2.1.4 Jacobi

La méthode de Jacobi de la librairie a été basée sur les équations du livre *Numerical Recipes* présentées ci-dessous, puisqu'elles utilisent le calcul matriciel, qui peut facilement être optimisé:

$$A' = (P_{pq})^T \bullet A \bullet P_{pq}$$

$$V = P_1 \bullet P_2 \bullet P_3 \ldots$$

$$D = V^T \bullet A \bullet V$$

où *A* est la matrice en entrée et *P* est la matrice de rotation générée par l'algorithme.

De plus, afin de trouver l'angle servant à créer les matrices de rotation de Jacobi, ce sont les équations du cours de Jim Lambers présenté à Stanford [4] qui ont été retenues, parce qu'elles prennent en compte l'instabilité numérique de la machine.

Aussi, la librairie fait preuve de plusieurs optimisations tirées du livre *Numerical Recipes* [1] qui permettent d'adapter l'algorithme pour être plus efficace en fonction de la précision numérique de la machine sur laquelle il est exécuté. Finalement, les versions cyclique et traditionnelle ont été implémentées afin de permettre à l'utilisateur d'adapter l'algorithme en fonction de ses besoins.

2.1.5 Descente de gradient

L'algorithme de la descente du gradient conjuguée utilisé se base sur les équations trouvées dans le livre *Numerical Recipes* [1].

Cet algorithme tient compte de deux vecteurs. Le gradient de la fonction au point que nous observons, soit le vecteur g. Le vecteur conjugué pour le point que nous observons, soit le vecteur h. L'algorithme consiste à obtenir le vecteur conjugué à partir du gradient à chaque itération. Le vecteur conjugué s'obtient grâce au calcul d'un facteur d'ajustement γ que l'on applique au vecteur g. Par la suite, on utilise une technique de recherche linéaire afin de trouver le minimum de la fonction dans la direction donnée par le vecteur conjugué. On se déplace ensuite à ce point et on recommence l'algorithme jusqu'à ce que le déplacement vers le nouveau point soit inférieur à la tolérance spécifiée par l'utilisateur, ce qui indique que nous avons trouvé un minimum.

La méthode de recherche linéaire utilisée est la méthode de Brent [6] décrite dans le livre *Numerical Recipes* [1]. Celle-ci utilisé un mixte d'interpolation quadratique et de recherche par bissection afin de trouver le minimum de la fonction dans la direction du conjugué.

Pour ce qui est de la méthode des différences finies utilisée pour calculer automatiquement le gradient si nécessaire, les équations décrivant les approximations de premier et second degrés sont les suivantes.

$$Premier\ degré: f'(x) = \frac{f(x+h) - f(x)}{h}$$

$$Second\ degré: f'(x) = \frac{f(x+h) - f(x-h)}{2h}$$

Dans les deux approximations, il est nécessaire de choisir une valeur de h petite afin d'avoir la meilleure approximation possible. L'utilisation de l'une ou l'autre de ces approximations dépend de la forme de la fonction que l'on veut optimiser et le choix sera donc à l'utilisateur de spécifier laquelle prendre.

2.1.6 Algorithme génétique

Deux solutions ont été retenues pour l'algorithme génétique. Une version plus rapide qui consomme plus de mémoire et une solution plus lente qui en consomme moins. Ces

solutions ont été retenues parce que la librairie a été conçue pour être exécutable sur tous les microcontrôleurs. Les petits microcontrôleurs n'ont pas la mémoire nécessaire pour exécuter la première version de l'algorithme , donc la deuxième a été implémentée pour répondre à ce besoin. La seule différence entre les deux versions est que la première enregistre les aptitudes de chaque solution après qu'elle ne soit calculée qu'une seule fois, tandis que la deuxième version recalcule cette valeur chaque fois qu'elle en a besoin lors de la sélection des parents par tournoi.

2.1.7 Algorithme de Monte Carlo

L'algorithme de Monte Carlo se divise en 4 étapes répétées de manières itératives jusqu'à ce que l'on atteigne une condition d'arrêt spécifiée par l'utilisateur .

La première étape est celle de la sélection du prochain nœud à visiter, qui sera alors le départ exploratoire. Cette sélection se fait en parcourant l'arbre d'actions en choisissant celle avec la valeur obtenue la plus élevée jusqu'à ce qu'on arrive à un nœud feuille de l'arbre. Ce nœud feuille est alors choisi comme départ exploratoire, duquel l'algorithme va simuler un épisode aléatoire.

La deuxième étape de l'algorithme est celle de l'expansion du nœud sélectionné. Cette expansion ne fait que trouver les prochaines actions qui peuvent être réalisées à partir du noeud de départ et créer ses noeuds enfants à partir de ces actions.

La troisième étape de l'algorithme est celle de la simulation. À partir du nœud sélectionné, il y a une simulation aléatoire de l'épisode jusqu'à un nœud terminal. Rendu à ce nœud terminal, le résultat de cet épisode est calculé.

La dernière étape est alors de propager le résultat obtenu de la simulation jusqu'au nœud racine de l'arbre et en incrémentant le nombre de visites de chaque nœud parcouru lors de la simulation.

La condition d'arrêt de cette boucle à 4 étapes peut être une contrainte du nombre d'itérations et/ou un seuil acceptable pour le résultat obtenu.

2.2 Quelques diagrammes de classes ou de modules

Par rapport aux modules de la librairie, il est important de mentionner que le client insiste sur le fait que la librairie ne doit dépendre d'aucune librairie externe. Seule la librairie standard de C peut être utilisée pour développer les algorithmes. C'est pourquoi, pour l'instant, la majorité des algorithmes développés sont indépendants entre les modules de la classe eux-mêmes. La structure des modules est décrite dans le diagramme suivant où les flèches représentent les dépendances entre ces derniers :

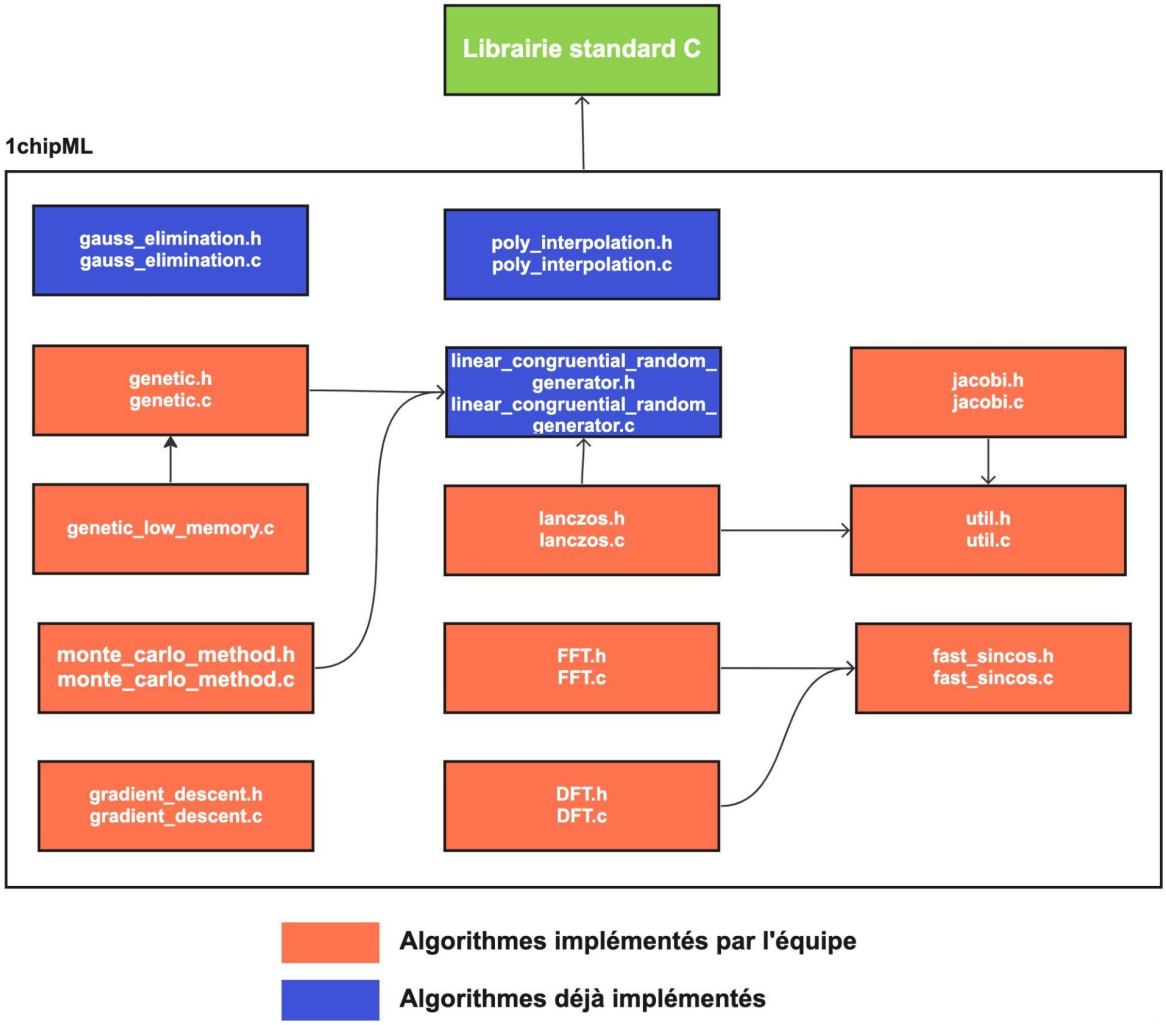

Figure 2 : Diagramme des modules de la librairie 1chipML

Aussi, le seul module dans la librairie qui n'est pas un algorithme à proprement parler est le module *util*. Ce module contient plusieurs méthodes qui sont utilisées plusieurs fois par les algorithmes comme la multiplication de matrice, la transposée de matrice et la mise à l'échelle de vecteurs.

Puisque la librairie est écrite en C, le concept de classe est inexistant. De plus, aucune structure spéciale n'est utilisée pour implémenter les algorithmes. Seuls les types fournis par le langage ont été utilisés lors du développement. Il n'est donc pas possible de fournir un diagramme de classes.

2.3 Diagrammes d'états et/ou d'interface usager

Puisque la tâche à compléter pour notre client est la création d'une librairie implémentant divers algorithmes mathématiques sur un microcontrôleur. La notion d'état et d'interface usager ne s'applique pas à notre projet. En effet, chacun de nos algorithmes est implémenté par une fonction qui s'exécute de manière séquentielle et

retourne une réponse. Il n'y a donc pas de notion d'état dans ceux-ci. Ensuite, puisque notre projet consiste en la création d'une librairie, nous fournissons des interfaces logicielles, c'est-à-dire des fonctions qui peuvent être appelées par les utilisateurs de la librairie. La notion d'interface usager ne s'applique donc pas non plus.

2.4 Interfaces logicielles et/ou matérielles importantes

Du côté logiciel, les interfaces importantes pour le développement de notre projet sont les diverses définitions des algorithmes que nous avons à implémenter.

Premièrement, chaque algorithme doit utiliser un type défini spécifiquement pour celui-ci. Par exemple, au lieu d'utiliser le type « double », chaque algorithme doit définir son propre type équivalent comme « lanczos_real » pour l'algorithme de Lanczos. Cela est fait afin d'éviter des problèmes reliés à la différence de taille et de précision de certains types tels que « float » et « double » sur différent modèle de microcontrôleur.

Ensuite, chaque algorithme développé expose aux utilisateurs de la librairie une fonction afin d'utiliser l'algorithme. Les paramètres de ces fonctions sont variables et dépendent du type d'algorithme qui est implémenté.

Pour les algorithmes de FFT et de DFT, ceux-ci permettent de transformer des données du domaine temporel au domaine fréquentiel. Les fonctions prennent en paramètre deux tableaux dont la taille est une puissance de deux, le premier étant la composante réelle des données et le second, leur composante complexe. Le résultat de l'algorithme sera renvoyé à l'utilisateur à travers ces deux mêmes tableaux.

Pour l'algorithme de Lanczos, celui-ci permet d'appliquer l'algorithme sur une matrice symétrique passé en paramètre d'entrée. Ensuite, l'algorithme décompose cette matrice en deux matrices T et V qui sont retournées à l'utilisateur.

Pour l'algorithme de Jacobi, celui-ci permet de déterminer les valeurs et vecteurs propres d'une matrice fournie en entrée. La matrice d'entrée doit être carrée et symétrique. Les valeurs propres et vecteurs propres seront retournés à l'utilisateur dans deux matrices distinctes.

Pour l'algorithme Monte Carlo, celui-ci permet d'itérer à travers un arbre d'actions pour déterminer la meilleure action à poser dans un contexte particulier. L'utilisateur doit donc fournir ce contexte et les actions qui peuvent en découler. Plus précisément, l'utilisateur doit décrire le jeu auquel il souhaite appliquer l'algorithme de Monte Carlo. Ces précisions sont les spécifications quant au plateau de jeu, les différentes actions possibles du jeu, l'exécution de l'action, le retrait de l'action et le résultat obtenu du jeu. De plus, l'utilisateur doit spécifier les conditions d'arrêts de l'algorithme, soit le seuil acceptable du résultat obtenu de l'algorithme, ou bien le nombre d'itérations maximal et minimal.

Pour l'algorithme génétique, celui-ci permet d'optimiser une fonction fournie en paramètres et retourne les valeurs optimales de celle-ci. Il est possible de spécifier plus exactement le comportement de l'algorithme à l'aide d'autres paramètres comme la taille des générations, la chance de mutations, etc.

Pour l'algorithme de la descente du gradient, celui-ci permet d'optimiser une fonction fournie en paramètre et retourne le minimum de celle-ci. Certains paramètres supplémentaires sont fournis à la fonction afin de spécifier la précision de celui-ci et pour limiter le nombre d'opérations.

Pour ce qui est des interfaces matérielles, celles-ci incluent des microcontrôleurs Arduino qui seront utilisés afin de créer des prototypes démontrant certaines applications réelles des algorithmes sur un microcontrôleur. Il sera donc nécessaire d'utiliser une communication sérielle entre l'Arduino et un ordinateur afin de récupérer les résultats produits par le Arduino lors de l'exécution des algorithmes.

2.5 Simulation, norme ou modèles importants

L'équipe ne fait appel à aucun logiciel de simulation dans le cadre du projet. Le projet est de développer une librairie d'algorithme pour faciliter l'implémentation de l'informatique de périphérie sur des microcontrôleurs.

L'équipe a toutefois mis en place des normes de programmation. La norme utilisée est le standard de programmation de C proposé par le groupe de développement LLVM. Ce standard est ensuite appliqué par le biais de l'exécutable « clang-format », un programme de formatage de fichier. Le standard se trouve à l'Annexe 1 de ce rapport. Il comporte plusieurs éléments importants, dont la ligne « AlignTrailingComments: true » qui ajuste les commentaires en fin de ligne pour qu'ils soient tous alignés horizontalement.

Un autre élément important de ce fichier est « ColumnLimit : 80 » , cette configuration force la taille maximale d'une ligne de nos fichiers à 80 caractères. Les lignes qui dépassent cette valeur vont être séparées. On force la taille des indentations à deux espaces avec la ligne suivante « IndentWidth: 2 ». Toutes ces modifications augmentent l'uniformité de nos fichiers de code et les rendent beaucoup plus lisibles.

Un autre standard créé par l'équipe est le format des commentaires présents dans les fichiers. Tous les commentaires de fonctions commencent par une brève description de l'objectif de la fonction, suivi par une description de chaque paramètre que nécessite la fonction.

Une autre norme qui a été décidée par le client est d'utiliser la version C99 du langage de programmation C. Ce choix a été fait parce que cette version est supportée sur la vaste majorité des compilateurs qui existent pour les microcontrôleurs.

Pour ce qui est des modèles, des prototypes ont été développés utilisant chacun des algorithmes dans la librairie. La section 2.7 de ce rapport explique les prototypes pour chaque algorithme et la section 3.2 explique la division de ces tâches.

2.6 Description d'outils, librairies ou cadre de travail (*framework*) utilisés (Q5.1)

Comme mentionné précédemment, la librairie n'a aucune dépendance externe. Par conséquent, son installation ne requiert que les fichiers fournis par le projet. Cependant,

lors du développement, plusieurs outils ont été utilisés pour s'assurer de son fonctionnement.

Tout d'abord, VSCode a été l'IDE principal lors du développement de la librairie pour sa facilité d'utilisation et sa personnalisation. De plus, les compilateurs « gcc » et « clang » ont été utilisés pour compiler les algorithmes et les tester sur des CPUs. Le fait de tester les algorithmes sur des CPUs permet de vérifier la logique des algorithmes sans se soucier des problèmes d'implémentation spécifiques aux MCUs. De plus, la compilation et l'édition de liens se font à l'aide de Makefiles.

Puis, par rapport à la compilation pour des microcontrôleurs, le compilateur « avr-gcc » est utilisé pour générer les programmes qui rouleront sur les Arduinos. Cependant, afin de faciliter l'utilisation de la librairie à ses futurs utilisateurs, c'est l'environnement de développement Arduino qui est utilisé lors de la compilation pour des MCUs. En effet, un des requis principaux du client est la simplicité d'utilisation de la librairie. Cependant, considérant que la compilation vers un système embarqué et la programmation d'un microcontrôleur varie selon plusieurs paramètres (famille de microcontrôleur utilisée, port de connexion, etc.), créer un Makefile pour gérer le tout compliquerait beaucoup la tâche aux utilisateurs de la librairie. En effet, ces derniers devraient spécifier lors de l'exécution du Makefile plusieurs paramètres en ligne de commande, ce qui devient redondant si ces derniers ne veulent qu'exécuter un simple algorithme. Par conséquent, lors du développement de prototypes d'utilisation des algorithmes, l'environnement de développement d'Arduino (Arduino IDE) est utilisé. C'est pourquoi le dossier *prototype* contient des fichiers *.ino*. Ces fichiers sont utilisés par Arduino IDE pour configurer la compilation et la programmation d'un microcontrôleur spécifique. Son interface usager facilite grandement le développement embarqué et le fichier a été configuré pour que les fichiers dans la librairie nécessaires aux prototypes soient importés directement. De plus, la détection de port se fait automatiquement et le choix du microcontrôleur est intuitif.

Puis, par rapport au standard de programmation, l'outil « clang-format » a été choisi par l'équipe pour adopter un style de codage uniforme dans toute la librairie.

En ce qui concerne la communication dans l'équipe et la gestion de version, les outils GitHub et Slack sont utilisés. GitHub permet notamment de gérer les conflits entre les différentes branches de code et de gérer plusieurs versions de la librairie en même temps. Puis, Slack permet d'établir une communication constante entre les membres de l'équipe et le client.

Afin de rédiger la documentation, un compilateur LaTeX est utilisé pour générer le fichier « pdf » final téléversé dans la librairie.

2.7 Quelques méthodes de test à appliquer pour valider la solution. (Q4.4)

2.7.1 Transformée de Fourier rapide (FFT) et transformée de Fourier discrète (DFT)

Afin de valider le fonctionnement de la DFT et de la FFT, une matrice et son résultat ont été calculés à l'avance et sont fournis en entrée aux deux algorithmes. Il est possible de déterminer si les algorithmes fonctionnent si leur résultat correspond au résultat

attendu. De plus, la DFT et la FFT sont utilisées pour vérifier le fonctionnement de l'un et de l'autre, à partir de valeurs aléatoires. En d'autres mots, des données générées aléatoirement sont envoyées vers les deux algorithmes et leurs résultats sont ensuite comparés. Étant donné qu'ils effectuent les mêmes types de calculs, leurs résultats devraient être pareils. Également, le prototype de convolution d'image avec FFT servira à valider le fonctionnement de l'algorithme sur microcontrôleur. En effet, des données seront envoyées vers un Arduino afin qu'il puisse appliquer une FFT sur celle-ci. Ensuite, les résultats sont renvoyés à l'utilisateur et seront utilisés pour analyser le résultat de la convolution.

2.7.2 Descente du gradient

Pour ce qui est de la méthode de la descente du gradient conjuguée. Afin de valider son bon fonctionnement, un premier test simple consiste à demander à la fonction de déterminer le minimum d'une fonction simple telle que $f(x, y) = (x - 4)^2 + (y + 3)^2$ et de voir si l'algorithme nous retourne le bon minimum de (4, -3) selon la tolérance spécifiée. Un second test à effectuer serait d'utiliser la descente du gradient sur le Arduino afin de déterminer le polynôme de degré N qui décrit un ensemble de points pour par la suite extrapoler ces points dans le futur. L'information calculée par le Arduino pourrait ensuite être envoyée à un ordinateur ou un programme python ce chargerait de valider celle-ci en effectuant lui aussi une régression polynomiale et comparait les résultats.

2.7.3 Génétique

Le premier test de l'algorithme génétique est très similaire à celui de la descente du gradient. Cet algorithme est utilisé pour trouver l'optimum global d'une équation mathématique. Ainsi, pour valider le fonctionnement de celui-ci, la fonction suivante est fournie en entrée: $f(x, y) = abs(4x - e^{(2*y)} + 2)$.

Pour le test, les valeurs de x et y sont bornées entre 0 et 1 pour simplifier le test. On vérifie ensuite que l'algorithme retourne une solution qui est valide selon la tolérance. Puisque cet algorithme n'a pas la garantie de trouver la solution exacte. La tolérance est requise pour vérifier que la solution retournée est suffisamment proche

Le deuxième test sera accompli directement sur le microcontrôleur. L'algorithme devra être en mesure d'utiliser une fonction polynomiale générique pour représenter un nuage de points et doit trouver les paramètres qui font correspondre la « spline » à la série de points. La méthode de validation sera la même que celle de la descente du gradient, nous utiliserons un programme python pour valider nos résultats à l'aide d'une régression polynomiale.

2.7.4 Lanczos

Pour valider le bon fonctionnement de l'algorithme de Lanczos, 2 tests seront développés et intégrés au projet.

Le premier test consiste en une vérification formelle de l'algorithme. Pour ce faire, une matrice a été sélectionnée et les résultats attendus de l'algorithme ont été calculés manuellement. Le test consiste donc à comparer les résultats obtenus suite à l'exécution de l'algorithme à ceux qui ont été précédemment calculés.

Le second test est aussi utilisé pour valider l'algorithme qui a été développé, mais est utilisé comme prototype pouvant être exécuté sur microcontrôleur. Ce test consiste à appliquer une analyse en composante principale (PCA) sur un jeu de données utilisé pour entraîner un modèle à effectuer de la classification. Suite à l'application de la PCA, ces données peuvent par la suite être représentées dans un espace en 2 dimensions tout en gardant la majorité des informations de l'espace initial.

2.7.5 Jacobi

Pour valider le fonctionnement de l'algorithme de Jacobi, deux tests ont été conçus.

Tout d'abord, le premier test consiste à vérifier qu'étant donné une matrice quelconque dont les valeurs et vecteurs propres sont connus, l'algorithme retourne le bon résultat. Il est important de mentionner que ce test est validé dans le cas de la version cyclique de l'algorithme et dans le cas de la méthode traditionnelle.

Le deuxième test consiste à effectuer une réduction de dimensionnalité pour compresser une image à l'aide d'une PCA. En effet, en prenant la matrice de covariance d'une image et en calculant les valeurs et vecteurs propres de celle-ci, il est possible d'enlever une grande partie des données sans compromettre l'information qui se trouve dans l'image. La matrice de covariance est envoyée à un microcontrôleur qui exécute l'algorithme de Jacobi pour ensuite renvoyer la réponse à un ordinateur qui affiche l'image. L'image est ensuite validée qu'elle a bien été compressée.

2.7.6 Monte Carlo

Pour valider le fonctionnement de l'algorithme Monte Carlo, deux tests sont implémentés. Le premier test est celui du jeu « Tic-Tac-Toe ». Il consiste en l'implémentation du jeu et des différentes fonctions qui sont spécifiques à celui-ci, comme vu dans la section 2.4. Ces fonctions sont passées en paramètres à celle exécutant l'algorithme Monte Carlo. Ensuite, le test vérifie si à partir d'un état connu du jeu « Tic-Tac-Toe » et dont le prochain état favorable est connu également, le résultat obtenu de l'exécution de l'algorithme est celui attendu. Un second test pour vérifier le bon fonctionnement de l'algorithme Monte Carlo est celui de la recherche du plus court chemin pour parcourir un graphe. Il consistera donc à passer à la fonction la description d'un graphe ainsi que les fonctions décrivant les différentes actions du parcours du graphe. Ensuite, il vérifiera si le parcours retourné est celui le plus court en comparant avec la valeur juste, connue d'avance.

3. Gestion du projet

3.1 Identification des tâches principales du projet (Q3.3)

Le projet consiste en l'implémentation d'algorithmes communs en intelligence artificielle à des fins d'utilisation sur des microcontrôleurs dans une librairie à code ouvert.

Le projet se divise donc en plusieurs tâches indépendantes entre elles, soit l'implémentation des différents algorithmes, et ensuite en sous-tâches dépendantes spécifiques à chaque algorithme, comme les tests, la documentation, les prototypes et l'optimisation.

3.1.1 Implémentation

Pour ce qui est de l'implémentation des algorithmes, voici une liste détaillée des tâches principales:

- Implémentation des algorithmes:
  - Transformée de Fourier rapide
  - Jacobi
  - Lanczos
  - Génétique
  - Descente du gradient
  - Monte Carlo pour l'apprentissage par renforcement

Après l'implémentation de ces algorithmes, d'autres algorithmes seront implémentés, tels que:

- « Multi-armed bandit »
- Des méthodes statistiques (analyse variance, covariance)
- Décomposition LU
- Méthodes des différences finies

Pour chacune des tâches définies dans les listes ci-dessus, il faut implémenter l'algorithme pour que ce dernier puisse s'exécuter sur un microcontrôleur.

3.1.2 Recherche

L'implémentation des algorithmes nécessite avant tout une bonne compréhension de ceux-ci. Puisque ces algorithmes n'étaient pas maîtrisés préalablement au projet, une tâche importante à réaliser pour ce dernier était celle de recherche sur les algorithmes à implémenter. Après s'être renseigné davantage sur la théorie derrière ceux-ci, une seconde tâche importante de recherche est celle sur les différentes implémentations possibles des algorithmes et leurs variantes. Ces tâches de recherche permettront d'être suffisamment outillés pour entamer les tâches d'implémentation des algorithmes. Voici donc une liste des tâches de recherche qui constituent le projet:

- Recherche sur les notions théoriques des différents algorithmes.
- Recherche sur les différentes implémentations des algorithmes et leurs variantes.

3.1.3 Optimisation

Après l'implémentation des algorithmes listés plus haut, puisque ces derniers sont destinés à s'exécuter sur des microcontrôleurs, des contraintes de mémoire doivent être prises en considération. Cela dit, une tâche importante à réaliser dans ce projet est l'optimisation de la mémoire utilisée par chaque algorithme. De plus, afin de rendre les algorithmes d'intelligence artificielle plus performants, un facteur important est celui du temps, il y a donc l'optimisation de la complexité de temps qui s'ajoute également aux tâches importantes du projet.

Voici donc une liste des tâches d'optimisation:

- Optimisation de la complexité d'espace pour chaque algorithme
- Optimisation de la complexité de temps pour chaque algorithme

3.1.4 Prototype

Une tâche importante du projet est celle de réaliser des prototypes pour chaque algorithme. Effectivement, les prototypes permettent aux utilisateurs de la librairie de mieux comprendre les applications de celle-ci, ainsi que de montrer l'utilité de celle-ci. Les prototypes sont donc des exemples d'application concrets des algorithmes. Ces prototypes doivent pouvoir rouler sur un microcontrôleur.

Voici donc une liste détaillée des tâches de prototypage ainsi que les différents algorithmes associés:

- Apprendre le jeu «Tic-Tac-Toe» (Monte Carlo pour l'apprentissage par renforcement)
- Parcours de graphe pour trouver le plus court chemin (Monte Carlo pour l'apprentissage par renforcement)
- Convolution d'image (Transformée de Fourier rapide)
- L'analyse d'images en composantes principales (Méthode de Jacobi et de Lanczos)
- Courbe d'ajustement (Algorithme génétique et descente du gradient)

3.1.5 Documentation

Une tâche importante du projet est celle de la documentation de chaque algorithme implémenté. Effectivement, puisque c'est un projet à code ouvert et qu'il aura une continuité à l'extérieur de notre cours, il est très important de documenter ce dernier de manière claire et uniforme. Non seulement est-il nécessaire de documenter les implémentations des algorithmes, les prototypes développés doivent l'être aussi. En effet, ces prototypes servent à la compréhension de la communauté du projet à code ouvert et rendent cette librairie plus simple d'usage. La documentation consiste en la génération d'un fichier .pdf avec la description des algorithmes, de leur implémentation et des méthodes d'utilisation de ceux-ci. La forme de la documentation est décrite davantage dans la section 2.5 du présent rapport.

S'ajoutent donc aux tâches, celle de la documentation:

- Documentation des implémentations des algorithmes
- Documentation des prototypes pour chaque algorithme

3.2 Répartition des tâches et responsabilités dans l'équipe

Nous avons commencé le projet en attribuant des rôles à chaque membre de l'équipe pour faciliter le déroulement du projet.

Florence est responsable des communications ainsi que de prendre des notes lors des réunions, c'est-à-dire qu'elle s'occupe de toute communication qui se déroule entre notre équipe.

Nicholas a le rôle de gestionnaire de l'environnement de travail. Il a créé et gère un canal Slack pour simplifier la communication avec le client lorsque des membres de l'équipe ont des interrogations sur la méthode d'accomplissement des tâches. Le deuxième élément de ce rôle était d'implémenter un standard de programmation décidé par l'équipe.

Paul-André a le rôle de gestionnaire des tests. L'équipe d'Ericsson qui contribue à ce projet a mis en place quelques tests qui s'exécutent par le biais d'un Makefile. Il est donc responsable de mettre une procédure uniforme pour le développement des tests. Cela augmente grandement la facilité de compréhension des tests par les tiers puisque la structure de chaque fichier de test est la même. Un utilisateur de la libraire peut alors facilement modifier les tests en fonction de leurs besoins.

Félix-Antoine est le responsable des requêtes de fusion. Tous les membres de l'équipe participent à la revue de code, mais il a la responsabilité finale de l'approbation de la requête de fusion ainsi que de s'assurer que le client est au courant de l'ouverture de la requête. Il est aussi responsable de faire un suivi des requêtes pour s'assurer que celles-ci ne bloquent pas.

Alexis est l'animateur, c'est-à-dire qu'il recueille les idées de tout le monde afin de créer les plans de rencontre pour nos réunions hebdomadaires en équipe. Il s'occupe de gérer nos réunions afin que tous les points soient abordés. Lors de nos rencontres avec le client, il s'occupe de partager les idées de l'équipe et de poser nos questions. Son rôle est très utile pour faciliter le déroulement des rencontres.

Gaya est le responsable du temps et du déroulement des tâches. Il s'assure que le temps pris par chaque point de notre plan de rencontre ne dépasse pas le temps qui lui est alloué. On minimise alors les chances de débordement de rencontre. Il est également le responsable de la gestion de notre tableau Kanban. Il s'assure que l'échéancier est respecté et modifie le tableau s'il y a des imprévus lors de l'accomplissement des tâches ou si le client a une demande supplémentaire.

Le tableau suivant montre la division des tâches qui ont été énoncées à la section 3.1 entre les membres de l'équipe .

Membres de l'équipe	**Tâches à accomplir**
Florence	● Implémentation de l'algorithme Monte Carlo (100 heures) ● Apprendre le jeu Tic-Tac-Toe (Monte Carlo pour l'apprentissage par renforcement) (45 heures) ● Parcours de graphe pour trouver le plus court chemin (Monte Carlo pour l'apprentissage par renforcement) (40 heures)
Nicholas	● Implémentation de l'algorithme génétique (85 heures) ● Courbe d'ajustement (Algorithme génétique) (40 heures)
Paul-André	● Implémentation d'un algorithme de transformée de Fourier rapide (85 heures) ● Convolution d'image (Transformée de Fourier rapide) (45 heures)

Félix-Antoine	● Implémentation d'un algorithme de Lanczos (95 heures) ● L'analyse d'images en composantes principales (Lanczos) (45 heures)
Alexis	● Implémentation de la technique d'optimisation de descente du gradient (85 heures) ● Courbe d'ajustement (Descente du gradient) (40 heures)
Gaya	● Implémentation de l'algorithme de Jacobi (95 heures) ● L'analyse d'images en composantes principales (Jacobi) (45 heures)
Tous les membres	● Recherche sur les notions théoriques des différents algorithmes. (5 heures par membre) ● Recherche sur les différentes implémentations des algorithmes et leurs variantes. (10 heures par membre) ● Optimisation de la complexité d'espace pour chaque algorithme implémenté (10 heures par membre) ● Optimisation de la complexité de temps pour chaque algorithme implémenté (10 heures par membre) ● «Multi-armed bandit» (80 heures) ● Des méthodes statistiques (analyse variance, covariance) (100 heures) ● Algorithme de d'algorithme de décomposition LU (80 heures) ● Méthodes des différences finies (80 heures) ● Documentation des implémentations des algorithmes (5 heures par membre) ● Documentation des prototypes pour chaque algorithme (10 heures par membre) ● Rencontres avec le client (10 heures par membre)

Tableau 1: Tâches à accomplir pour chaque membre de l'équipe

3.3 Mesure sommaire de la progression du projet

Afin d'évaluer la progression du projet, nous avons décidé d'évaluer individuellement la complétion de chacun des six premiers algorithmes que nous devions implémenter. Pour chacun des algorithmes, nous avons estimé le travail effectué et le travail restant de la manière suivante. 20% du travail est associé à l'activité de recherche de la variante de l'algorithme à utiliser. 45% du travail est associé à l'implémentation de l'algorithme. 5% du travail est associé à nettoyer et formater le code ainsi qu'à répondre aux divers commentaires dans les requêtes de fusion. 5% du travail est associé à la

création d'un test simple pour valider le bon fonctionnement de l'algorithme. 15% du travail est associé à l'implémentation d'un prototype pour l'algorithme avec le Arduino. 10% du travail est associé avec la rédaction de la documentation associée à l'algorithme.

Le tableau suivant indique le pourcentage de complétion de chacun des algorithmes. Il est important de noter que dans celui-ci chaque algorithme possède le même poids que les autres.

Algorithmes	Complétion	Tâches restantes
FFT/DFT	85%	● Implémentation d'un prototype sur le Arduino ● Optimisation pour Arduino
Jacobi	85%	● Implémentation d'un prototype sur le Arduino ● Optimisation pour Arduino
Lanczos	80%	● Implémentation d'un prototype sur le Arduino ● Optimisation pour Arduino ● Nettoyage et Formatage
Algorithme Génétique	80%	● Implémentation d'un prototype sur le Arduino ● Optimisation pour Arduino ● Nettoyage et Formatage
Descente du gradient	80%	● Implémentation d'un prototype sur le Arduino ● Optimisation pour Arduino ● Nettoyage et Formatage
Algorithme de Monte Carlo	80%	● Implémentation d'un prototype sur le Arduino ● Optimisation pour Arduino ● Nettoyage et Formatage
Décomposition LU	0%	● Toutes les tâches nécessaires pour un algorithme
Méthodes des différences finies	0%	● Toutes les tâches nécessaires pour un algorithme

« Multi-armed bandit »	0%	● Toutes les tâches nécessaires pour un algorithme
Méthodes statistiques	0%	● Toutes les tâches nécessaires pour un algorithme

Tableau 2: Progrès des algorithmes en cours de développement

Grâce à ce tableau, on peut estimer le progrès global du projet pour les six algorithmes présentement en développement entre 80% et 85%. Considérant que l'on désire réaliser environ 10 algorithmes, on peut évaluer que la progression globale de notre projet se situe à 49%.

3.4 Mesures de révision du travail réalisé par les autres membres de l'équipe

Chaque algorithme devant être ajouté à la librairie 1chipML doit être développé en suivant un processus de validation permettant de s'assurer que celui-ci respecte les normes imposées par l'équipe du projet. Les choix d'implémentations sont aussi validés par l'ensemble de l'équipe et le client pour s'assurer de prendre les meilleures décisions quant aux algorithmes à implémenter. Ainsi, le processus de développement et d'intégration d'un algorithme est divisé en trois étapes.

En premier lieu, une recherche préliminaire doit être faite sur l'algorithme pour bien comprendre le comportement et les détails de l'implémentation de celui-ci. Lors de cette étape, les limitations de l'algorithme doivent être identifiées et une recherche des solutions doit être faite pour réussir à pallier ces limitations. Cette recherche préliminaire doit par la suite être présentée à l'équipe et au client pour discuter des solutions proposées et s'entendre sur les détails de l'implémentation qui doit être faite.

En second lieu, l'implémentation de la librairie doit être faite en C99 sur une branche différente de la branche principale du projet. Lorsque cet algorithme est fonctionnel, testé et documenté, une requête de fusion doit être ouverte sur l'entrepôt du projet et les changements doivent être revus par 2 membres de l'équipe. Lors de cette étape de validation, une vérification est effectuée au niveau du respect des standards de programmations définies par l'équipe. Une validation est aussi faite au niveau de la logique de l'algorithme implémentée, des performances obtenues ainsi que des tests implémentés pour valider l'algorithme. Les membres de l'équipe peuvent soit approuver la requête de fusion ou demander des changements supplémentaires. Si des modifications sont demandées, la personne responsable du développement de

l'algorithme à la responsabilité de mettre à jour son implémentation en fonction de ce qui a été demandé. Si, cependant, les changements demandés ne sont pas applicables, une explication doit être fournie par l'auteur du code pour expliquer pourquoi les changements demandés ne peuvent pas être appliqués. Ce processus est fait de manière itérative plusieurs fois jusqu'à ce que les membres responsables de la révision aient tous approuvé la requête de fusion.

Finalement, lorsque le code soumis a été révisé par certains membres de l'équipe et que ceux-ci considèrent que le code est complet et fonctionnel, le code est par la suite proposé au client pour une dernière validation. À la demande du client, le code est révisé en direct lors des réunions dans les bureaux d'Ericsson. Ainsi, lorsque le client considère que les changements proposés sont valides et respectent ces exigences, le code peut être intégré dans la branche principale du projet pour être rendu disponible officiellement à tous ceux qui désirent intégrer l'algorithme dans leur projet.

4. Progression jusqu'à la fin du projet et conclusion

4.1 Retour sommaire sur le travail déjà réalisé

Lors de la première moitié de la session, nous avons été en mesure d'implémenter une partie des algorithmes demandés par le client ainsi que des fonctionnalités connexes à chaque algorithme. La documentation a été mise à jour pour expliquer le fonctionnement de chaque algorithme ainsi que les limitations de ceux-ci. Finalement un ou plusieurs tests ont été mis en place pour vérifier que ceux-ci fonctionnent correctement. Plus exactement, 6 algorithmes proposés par le client au début de la session ont été implémentés et sont actuellement soit en processus de revue de code final par le client ou déjà intégré dans le dépôt officiel.

Ainsi, les algorithmes FFT et DFT ont tous les deux été développés pour permettre d'effectuer des transformées de Fourier sur une série de données. Des tests ont été implémentés sur les deux algorithmes pour permettre de vérifier que ceux-ci sont fonctionnels et les résultats sont comparés entre les deux algorithmes pour s'assurer que le résultat obtenu est le même puisque ceux-ci calculent la même information et les résultats ne devraient donc pas changer. De plus, la documentation a été mise à jour pour expliquer le fonctionnement de ces deux algorithmes et présenter les choix de conceptions.
Ensuite, l'algorithme de Lanczos a été implémenté pour permettre d'obtenir les valeurs propres et vecteurs propres d'une matrice auto-adjointe. La documentation a aussi été mise à jour pour présenter le fonctionnement de l'algorithme et expliquer son utilité. De plus, un test a été ajouté à la librairie pour vérifier que l'algorithme fonctionne

correctement. Dans ce test, une matrice a été calculée à la main et les résultats de l'algorithme sont comparés à la réponse calculée manuellement.
De plus, l'algorithme de Jacobi a aussi été implémenté pour permettre d'obtenir les valeurs propres et vecteurs propres d'une matrice symétrique. L'algorithme de Jacobi est habituellement utilisé avec l'algorithme de Lanczos pour permettre d'accélérer le calcul sur des matrices dispersées. L'algorithme a donc été implémenté et la documentation associée décrivant le fonctionnement de l'algorithme ainsi que son usage a été mise à jour. Finalement un test a aussi été ajouté pour valider que l'algorithme effectue correctement le travail demandé. De la même manière que l'algorithme de Lanczos, une matrice de base est donnée et une validation est faite sur les résultats obtenus. Ces résultats sont connus à l'avance puisqu'ils ont été calculés manuellement, une simple comparaison est donc suffisante.
Ensuite, une implémentation de l'algorithme génétique a été ajoutée à la librairie pour pouvoir trouver l'optimum de fonctions mathématiques. Puisque de nombreuses variantes de cet algorithme existent, certaines décisions ont été prises quant à l'implémentation et ces décisions sont expliquées dans la documentation associée. Un test a aussi été produit pour valider le fonctionnement de l'algorithme. Dans ce test, une fonction est donnée en entrée et nous nous assurons que l'optimum peut être trouvé par l'algorithme. Puisque l'algorithme génétique est majoritairement stochastique, une implémentation déterministe de la fonction aléatoire a été utilisée. Ceci permet de s'assurer que nous obtenons toujours les mêmes résultats d'une exécution à une autre. Ainsi, si un comportement aléatoire est détecté nous savons que celui-ci provient d'un comportement non défini plutôt que de la partie aléatoire de l'algorithme.
Ensuite, l'algorithme de descente de gradient a été implémenté. Similairement à l'algorithme génétique, cet algorithme permet d'optimiser une fonction mathématique. La documentation présente sur la librairie a été mise à jour pour expliquer le fonctionnement de l'algorithme ainsi que l'usage conseillé de celui-ci. Un test a aussi été ajouté à l'algorithme génétique. Une fonction mathématique est donnée en entrée et l'algorithme est exécuté de manière à optimiser les paramètres. Ce test permet donc de s'assurer que l'optimum peut être atteint et que l'algorithme fonctionne correctement.
Finalement, l'algorithme de Monte Carlo a aussi été développé et ajouté au projet. Cet algorithme est utilisé pour prédire quelle serait la meilleure action à prendre lorsqu'un événement incertain est présent. La documentation associée à cet algorithme a été mise à jour sur le dépôt du projet expliquant plus en détail le fonctionnement de l'algorithme, des limitations qui lui sont associées et une explication de comment intégrer cet algorithme dans un projet. Des tests ont aussi été ajoutés sur le jeu du «Tic-Tac-Toe» pour prédire les meilleurs coups à jouer sur certaines configurations précises du plateau. Encore une fois, la fonction aléatoire utilisée est déterministe pour s'assurer que les résultats obtenus lors des tests restent les mêmes d'une exécution à une autre.

4.2 Identifications des facteurs encore mal connus qui pourraient poser problème d'ici la fin du projet (Q2.6)

Un des facteurs le moins bien connus est celui associé à l'optimisation désirée par le client. Jusqu'à maintenant, les algorithmes ont été implémentés de sorte que les contraintes liées au matériel soient satisfaites. Cependant, après quelques discussions avec le client, il semble que d'autres optimisations plus poussées soient désirées. Par exemple, il a été mentionné quelques fois qu'il faut réduire le nombre de divisions pour certains cas, alors qu'il est acceptable d'en avoir pour d'autres. En apparence, ces optimisations peuvent paraître légères, mais certaines impliquent d'ajouter beaucoup d'effort et de temps. Un autre exemple est qu'il a été mentionné par le client que nous devons implémenter nous-mêmes nos propres fonctions sinus et cosinus, afin de les accélérer. Déjà là, il est possible de remarquer que cette nouvelle contrainte est une addition non négligeable au projet, comparativement à quoi il serait possible de s'attendre. En d'autres mots, les optimisations exactes désirées par le client ne sont pas encore claires et influencent grandement le temps requis pour le projet.

Un autre facteur imprévu s'agit des tests et prototypes que le client souhaite obtenir. Initialement, nous n'avons pas accès aux tests ni au type de prototype que le client souhaite obtenir pour chaque algorithme à implémenter. Il faut d'abord vérifier l'algorithme avec le client et c'est à ce moment que nous pouvons avoir plus de détails sur ses attentes quant à cet aspect. Il y a alors plusieurs facteurs inconnus lors du développement de chaque algorithme, pouvant faire en sorte que la conception soit plus longue. Néanmoins, le client nous donne tout de même différents choix de tests et de prototypes, nous laissant une certaine marge de manœuvre.

Finalement, un dernier facteur mal connu qui pourrait poser problème est l'implémentation des prototypes sur les Arduinos. Puisque les Arduinos ont été obtenus très récemment, il est possible que certaines méthodes utilisées pour les algorithmes posent problème en embarqué, mais pas sur l'ordinateur. L'obtention des Arduinos cinq semaines après le début du projet constitue un élément encore inconnu pour l'équipe, vu que nous avons peu expérimenté avec ce dernier. L'équipe risque alors de se retrouver dans des situations où plus de temps doit être passé sur les algorithmes afin de mieux les accommoder en embarqué et de permettre aux prototypes de bien fonctionner sur ceux-ci.

4.3 Principales tâches à réaliser pour assurer le succès du projet et que l'équipe estime pouvoir compléter avant la fin (Q3.1)

En comparant les tâches importantes du projet avec le travail déjà réalisé, voici les principales tâches qu'il reste à réaliser afin d'assurer le succès du projet.

4.3.1 Implémentation

Le succès du projet selon le client se mesure avec un minimum de 1 algorithme développé par personne. Cependant, l'équipe estime pouvoir compléter avant la fin du projet 8 à 10 algorithmes. En plus des algorithmes déjà implémentés par chacun, voici une liste de futurs algorithmes à implémenter avant la fin du projet:

- Implémentation des algorithmes:
    - « Multi-armed bandit »
    - Des méthodes statistiques (analyse variance, covariance)
    - Décomposition LU
    - Méthodes des différences finies

4.3.2 Exécution sur Arduino

De plus, pour compléter le projet, il est nécessaire de s'assurer que les algorithmes peuvent s'exécuter sur un microcontrôleur. S'ajoute donc aux tâches d'implémentation des algorithmes celle de preuve de concept sur les Arduinos qui sont mis à notre disposition, et cela pour tous les algorithmes.

- Preuve de concept sur Arduino (pour tous les algorithmes implémentés)

4.3.3 Optimisation

Pour chaque algorithme déjà implémenté, il reste la tâche d'optimisation de ceux-ci. L'optimisation sera d'ailleurs avantageuse pour permettre d'exécuter les algorithmes sur un Arduino.

De même, la tâche d'optimisation pour les nouvelles implémentations sera également nécessaire.

Voici donc une liste des tâches d'optimisation:

- Optimisation de la complexité d'espace pour les algorithmes
- Optimisation de la complexité de temps pour les algorithmes

4.3.4 Recherche

Comme pour les algorithmes déjà implémentés, une tâche importante avant d'entamer l'implémentation des prochains algorithmes de la liste ci-dessus est celle de la recherche. La recherche se divise de la même manière que pour celle réalisée pour les algorithmes passés: la recherche sur les notions théoriques des algorithmes ainsi que la recherche sur les différentes variantes d'implémentation de ceux-ci.

- Recherche sur les notions théoriques des différents algorithmes.
- Recherche sur les différentes implémentations des algorithmes et leurs variantes.

4.3.5 Prototypes

Il reste également la conception des différents prototypes pour chacun des algorithmes implémentés. Voici donc la liste des prototypes qu'il reste à implémenter et les algorithmes associés:

- Parcours de graphe pour trouver le plus court chemin (Monte Carlo pour l'apprentissage par renforcement)
- Convolution d'image (Transformée de Fourier rapide)
- L'analyse d'images en composantes principales (Méthode de Jacobi et de Lanczos)
- Courbe d'ajustement (Algorithme génétique et descente du gradient)

Concernant les prochains algorithmes qui seront implémentés lors du projet (énumérés plus haut dans la section *Implémentation)*, ceux-ci auront également des prototypes associés pour assurer leur bon fonctionnement. Cependant, ces prototypes n'ont pas encore été définis par le client.

4.3.6 Documentation

La tâche de documentation en est une constante avec l'ajout de nouveaux algorithmes et de prototypes. S'ajoutent donc aux tâches qu'il reste à faire pour la bonne complétion du projet, celle de la documentation:

- Documentation des algorithmes
- Documentation des prototypes pour chaque algorithme

4.3.7 Itérations avec le client

Pour assurer le succès du projet, il est nécessaire d'avoir une tâche d'itération avec le client. Cette tâche consiste en un retour sommaire avec le client sur ce qui a été effectué et sur ce qui est planifié pour la suite. Cette itération se fait lors des rencontres bimensuelles avec le client. Pendant celle-ci, nous repassons sur le code implémenté et nous discutons des objectifs pour l'itération qui suit.

Cette tâche est très importante pour s'assurer de la satisfaction du client avec le produit.

5. Références utilisées (Q3.2)

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, Second. Cambridge, USA: Cambridge University Press, 1992.

[2] Y. Saad, *Numerical methods for large eigenvalue problems*. Manchester, England: Society for Industrial and Applied Mathematics, 1992.

[3] C. Lanczos, *An iteration method for the solution of the Eigenvalue Problem of linear differential and integral operators*, Journal of Research of the national bureau of Standards, vol. 45, pp. 255–282, Oct. 1950.

[4] J. Lambers, *Advanced Topics in Numerical Linear Algebra*, Stanford, 2010. [En ligne]. Disponible : https://web.stanford.edu/class/cme335/lecture7.pdf

[5] “Finite Difference,” dans *Wikipédia*, 11 fév. 2023. [En ligne]. Disponible : https://en.wikipedia.org/w/index.php?title=Finite_difference&oldid=1138691183

[6] “Line Search,” dans *Wikipédia*, 20 juil. 2022. [En ligne]. Disponible : https://en.wikipedia.org/w/index.php?title=Line_search&oldid=1099362792

[7] Charbonneau, P. (2002). *An Introduction to Genetic Algorithms for Numerical Optimization* (No. NCAR/TN-450+IA). University Corporation for Atmospheric Research. doi:10.5065/D608638S

[8] Richard S. Sutton et Andrew G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, Massachusetts London, England: The MIT Press, 2014-2015. [En ligne]. Disponible: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

6. Annexe

Annexe 1 : Fichier de formatage .clang-format

```
---
Language:        Cpp
# BasedOnStyle:  LLVM
AlignAfterOpenBracket: Align
AlignOperands:   true
AlignTrailingComments: true
AllowAllArgumentsOnNextLine: true
AllowAllParametersOfDeclarationOnNextLine: true

AllowShortBlocksOnASingleLine: Never
AllowShortCaseLabelsOnASingleLine: false

AllowShortFunctionsOnASingleLine: All
AllowShortIfStatementsOnASingleLine: Never
AllowShortLoopsOnASingleLine: false
AlwaysBreakAfterReturnType: None
BraceWrapping:
  AfterCaseLabel:  false
  AfterControlStatement: false
  AfterEnum:       false
  AfterFunction:   false
  AfterStruct:     false
  AfterUnion:      false
  AfterExternBlock: false
  BeforeElse:      false
  IndentBraces:    false
  SplitEmptyFunction: true
  SplitEmptyRecord: true
BreakBeforeBinaryOperators: None
BreakBeforeBraces: Attach
BreakBeforeTernaryOperators: true
BreakStringLiterals: true
ColumnLimit:     80
ContinuationIndentWidth: 4
IncludeBlocks:   Merge
IndentCaseLabels: false
IndentWidth:     2
IndentWrappedFunctionNames: false
KeepEmptyLinesAtTheStartOfBlocks: true
```

```
MaxEmptyLinesToKeep: 1
PointerAlignment: Left
ReflowComments:  true
SortIncludes:    true
SpaceBeforeAssignmentOperators: true
SpaceBeforeParens: ControlStatements
SpaceInEmptyParentheses: false
SpacesBeforeTrailingComments: 1
SpacesInConditionalStatement: false
SpacesInContainerLiterals: false
SpacesInCStyleCastParentheses: false
SpacesInParentheses: false
SpacesInSquareBrackets: false
SpaceBeforeSquareBrackets: false
UseTab:          Never
…
```

Annexe 2 : Lien pour accéder au tag présent sur le répertoire GitHub du projet:

https://github.com/felix642/1chipML/tree/midterm