logoUHA imagePageDeGarde

Artificial Bee Colony Algorithm (ABC)

MUHIRWA GABO Oreste

GEILLER Valentin

Table des matières

Remerciement

Tout d'abord nous tenons à remercier particulièrement et à témoigner notre reconnaissance à notre professeur Monsieur Lhassane IDOUMGHAR pour nous avoir aidé à comprendre ce qu'était un métaheuristique.

Nous le remercions également de nous avoir confié ce projet qui a été un moyen pour nous de mettre en pratique les différentes notions des algorithmes évolutionnistes.

Par ailleurs, nous tenons également à remercier nos autres professeurs de la FST, qui nous ont permis d'acquérir des connaissances sur certains outils que nous avons eu à utiliser dans la concrétisation de ce projet.

Introduction

ABC (Artificial Bee Colony Algorithm)

L'algorithme de colonie d'abeilles est l'un des algorithmes (swarm), inspiré par le comportement collectif des colonies sociales d'insectes et d'autres sociétés animales.

Les propriétés du méthode SWARM

- Auto-organisation:
  → L'organisation d'une colonie est faite automatiquement, et cette intelligence de chaque individu collabore pour le résultat final.

- La division du travail:
  → Les tâches sont effectuées simultanément, et chaque travailleur a son rôle bien précis.

Cet algorithme modélise le comportement des abeilles à miel et peut servir à rechercher des solutions à des problèmes complexes. Les abeilles à miel jouent différents rôles dans leurs colonies, il y a des abeilles qui sont encore jeunes, et des abeilles adultes travailleuses. Parmi ces abeilles travailleuses, chaque abeilles doit exécuter une tâche bien précise. Ces abeilles travailleuses sont groupées dans 3 types principaux selon le type de tâche à accomplir.

- Employed Bees
- Onlooker Bees
- Scout Bees

Ces types d'abeilles résument aussi les étapes (phases) que notre algorithme doit dérouler pour arriver à résoudre les problèmes, ce qui est le but de ce projet.

Il y aura donc 3 phases principales, qui vont être détaillés dans les pages suivantes.

Dans ce document, il y aura des abréviations utilisées, notamment dans des formules mathématiques.

Abréviations	**Définitions**
fit	Fitness Value de la solution
fit_{new}	Fitness Value d'une nouvelle solution
X	Solution
X_{new}	Nouvelle Solution
f_i	Objective Value de la solution numéro i
f_{new}	Objective Value d'une nouvelle solution
$Prob_i$	Probabilité numéro i
O_i	Onlooker bee numéro i

E_i	Employed Bee numéro i
r	Un nombre aléatoire entre 0 et 1
ϕ	Un nombre aléatoire entre -1 et 1
X^j	j^{ème} variable de la solution actuelle
X^j_{new}	j^{ème} variable d'une nouvelle solution
X^j_p	J^{ème} variable de la p^{ème} solution

Sources de nourritures (Food source)

Pour que la source de nourriture puisse être utilisée, il ne faut pas qu'elle soit trop éloignée de la ruche, il faut qu'elle soit riche en nourriture, que ça soit facile d'extraire la nourriture. Ces paramètres sont à prendre en compte.

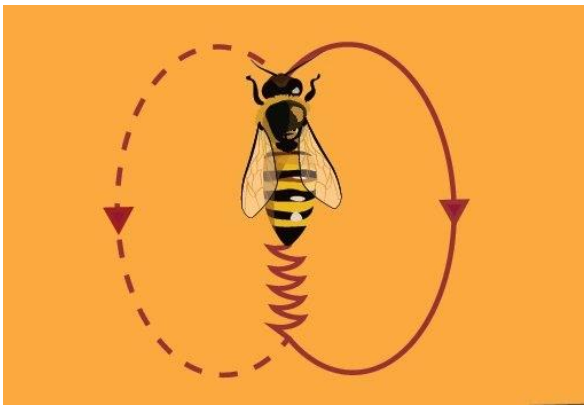
Butineuses actives

Les abeilles butineuses actives volent jusqu'à une source de nourriture, explorent les sources de nourriture voisine, recueillent la nourriture et reviennent à la ruche.

Ces abeilles doivent également partager entre eux, les informations comme la distance, et la direction pour que les autres puissent trouver la source de nourriture très facilement

La waggle danse

Une abeille exécute la waggle danse, quand elle veut informer les autres abeilles d'une source de nourriture qu'elle a trouvées, la danse se produit sur une piste située près de l'entrée et la sortie pour faciliter la sortie des abeilles butineuses.



Waggle Dance

Butineuses aux chômages

1. Onlookers

Il arrive que les ouvriers actifs deviennent inactifs. Dans ce cas, ils restent à l'entrée de la ruche, en attendant que les ouvriers actifs reviennent, une fois arrivés, les ouvriers actifs effectuent une danse qui sert à passer des informations au butineurs inactifs, et ces butineurs observent attentivement et ensuite ils peuvent à leur tour devenir actifs. Ils observent la danse des abeilles butineuses actives, pour bien saisir la direction et la distance à parcourir pour trouver la source de nourriture.

2. Scout

Avec certaines données fournies par les autres abeilles, ces abeilles savent quand il faut abandonner une source de nourriture, avec une méthode logique (plus détaillées à la page 8-9).

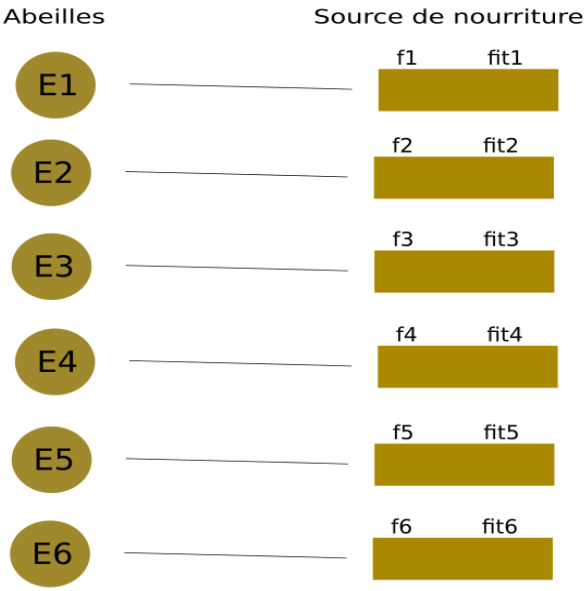
I. Présentation de l'algorithme

1. Employed Bee Phase

- Les abeilles employées tentent d'identifier une meilleure source de nourriture que celle qui lui est associée. Chaque abeilles donnera une nouvelle sources de nourritures.

- Générer une nouvelle solution à l'aide d'une solution partenaire.
- Greedy selection: cette étape consiste à mettre à jour, accepter une nouvelle source de nourriture.

Dans cette phase, chaque abeille doit exploiter une source de nourriture.



Chaque solution (source de nourriture) a un potentiel de créer une nouvelle source de nourriture.

Le nombre d'abeilles est égal au nombre de source de nourriture.

Une nouvelle solution est générée en modifiant une variable sélectionnée d'une façon aléatoire.

$$X_{new}^{j} = X^{j} + \phi * \left(X^{j} - X_{p}^{j}\right)$$

Exemple :

Avec $X_1 = [2 \quad 1 \quad 6 \quad 9]$, $X_2 = [0 \quad 4 \quad 7 \quad 2]$ et $j = 2$

Posons $\phi = -0.1$

$X_{new}^{2} = 1 + (-0.1) * (1 - 4) = 1.3$

$X_{new} = [2 \quad 1.3 \quad 6 \quad 9]$

2. Onlooker Bee Phase

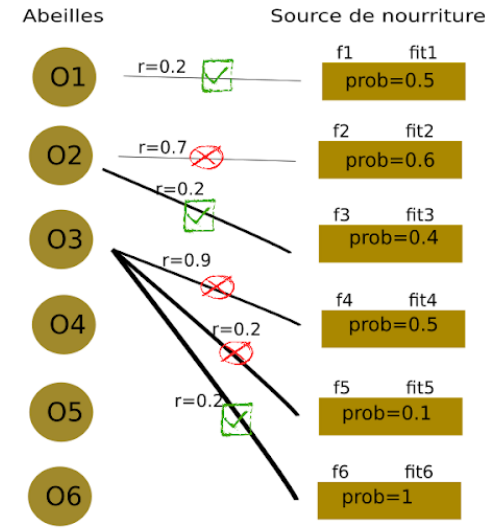
- À partir des informations communiquées par les abeilles actives dans l'étape employed bee phase, on tente d'identifier la meilleure source de nourritures, en tenant compte de la quantité et la qualité du nectar.
- Générer une nouvelle solution en se basant sur les probabilités

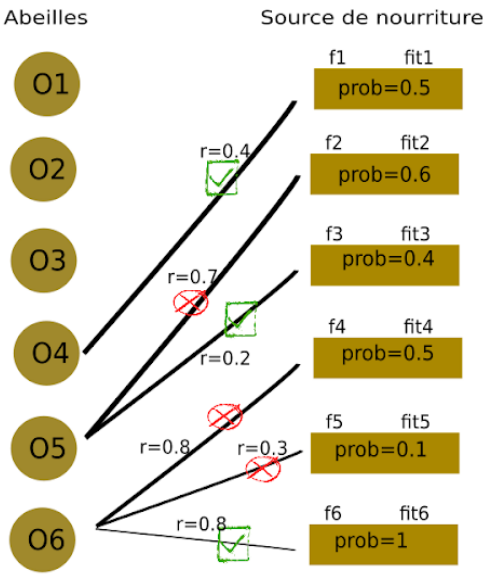
- Greedy selection: cette étape consiste à mettre à jour, accepter une nouvelle source de nourriture.

Pour calculer une probabilité, dans notre algorithme nous utilisons la formule suivante.

$$Prob_i = 0.9 * \left(\frac{Fitness}{maxFitness}\right) + 0.1$$

La condition pour qu'une solution soit générée ou non, est que notre nombre **r** soit inférieur à la probabilité de la source de nourriture actuelle.





Si la condition est respectée, une solution est générée et une liaison est créée.

$$r < Prob_i$$

Dans cette image, il y a 3 liaisons refusées, parce que le **r** généré aléatoirement est supérieur à la probabilité à la source de nourriture.

Dans ce genre de situation, l'abeille passe à la source de nourriture suivante et la même opération est répétée encore et encore, jusqu'à ce que la liaison devienne possible, ce qui est toujours le cas, car parmi les probabilités calculées, il y aura forcément une qui aura la valeur de 1 quand le fitness actuel est aussi le maximum.

Chaque fois que la condition est respectée et que la liaison créée,

- On génère une nouvelle solution
- On l'injecte dans la population si la solution est meilleure que celle actuelle
- Et on remet le nombre d'essai à zéro

Si la condition n'est pas respectée, on incrémente le nombre d'essai d'un.

Une fois toutes les sources de nourriture ont été parcourues, comme vous pouvez le voir sur les deux images, on recommence par la source de nourriture numéro 1.

Une remarque très importante, est que, contrairement à la phase précédente, là où chaque source de nourriture à le potentielle de générée une nouvelle source de nourritures meilleures qu'elle, ici cela dépend d'une probabilité calculée préalablement. Des sources ne peuvent ne pas générer une nouvelle source et donc il y en aura forcément qui ne seront pas exploitées.

3. Scout Bee Phase

- Les sources de nourritures épuisés sont abandonnées.
- Génère une nouvelle solution de manière aléatoire pour remplacer celle trouvée

Dans l'étape précédente, on avait une variable (comme un compteur) qu'on incrémente à chaque essai quand la condition nécessaire n'était pas satisfaite.

Avec une limite donnée donné par l'utilisateur, ou peut être calculé, dans cette phase, on doit éliminer une solution avec des conditions suivantes.

Les solutions avec le nombre d'essais supérieurs à la limite ont une chance d'être jetées.

Exemple : Posons une limite de 7.

On doit trouver la solution qui a le nombre d'échecs le plus élévé et qui en plus est supérieur à la limite. Nous pouvons avoir 3 cas possibles que voici.

F_i *Source de nourriture i*
SCB Scout Bee Phase

CAS 1	CAS 2	CAS 3
Une seule source de nourriture avec la condition.	Plusieurs sources de nourriture valident la condition. On prend la source avec le nombre d'échecs le plus haut.	Plusieurs sources de nourriture remplissent la condition. Plusieurs sources de nourriture ont le même nombre d'échecs. On choisit de manière aléatoire parmi les sources.
➔ F2 est choisi	➔ F2 est choisi	➔ F3 (ou F4) est choisi PAS LES DEUX.

On dit que la source de nourriture entre dans la scout bee phase.

Évaluation du fitness et la greedy selection

Calcule de la valeur Fitness	Greedy Selection
$fit = \begin{cases} \frac{1}{1+f} & Si\ f \geq 0 \\ 1+\lvert f \rvert & Si\ f < 0 \end{cases}$	$\left.\begin{matrix} X = X_{new} \\ f = f_{new} \end{matrix}\right\} si\ fit_{new} > fit$

II. Implémentation en C++

```
class OptUHA
{
protected:
	int   pop_size = 100;
	int   dimension = 30;
	int   total_func_evals = 5000;
	int   max_func_eval = total_func_evals * pop_size;
	double LOWER_LIMIT = -100;
	double UPPER_LIMIT = 100;
protected :
	double fct_obj(vector<double>& coords) const;
	double Sphere(const vector<double>& coords) const;
	double Rosenbrock(const vector<double>& S) const;
	double Rastrigin(const vector<double>& S) const;
	double Griewank(const vector<double>& S) const;
	double fitness(double objectifValue) const;
	vector<FoodSource> GenerateRandomPop();
	double generate_random_double();
	double generate_random_double(double lower, double upper);
	int generate_random_int(int from, int to);
	double newCoords(double X, double XPartner);
	FoodSource create_new_individual();
	void print_solution(const FoodSource& FS) const;
	void verifLimit(double& Xnew);
	bool greedySelection(double oldFit, double newFit);
	OptUHA() = default;
	~OptUHA() = default;
};
```

Notre classe OptUHA nous sert à générer la population et à l'évaluer. Elle permet aussi d'utiliser certaines fonctions utilitaires comme générer un nombre aléatoire, générer une probabilité, créer des nouveaux individus ou encore vérifier les limites du problème.

```
class myAlgorithm : OptUHA
{
protected:
	int NB_ABEILLES_EMPLOYE = pop_size / 2;
	int NB_ABEILLES_SPECTATEUR = pop_size / 2;
	int NB_FOOD_SOURCE = pop_size / 2;
	int LIMIT_TRIAL = NB_ABEILLES_EMPLOYE * dimension;
	vector<FoodSource> d_population;
	vector<double> d_probabilite;
	FoodSource d_bestFS;
public:
	double solve();
	myAlgorithm();
	~myAlgorithm() = default;
	void initializePopulation();
	double maxFitness() const;
	int indexOfSolutionWhereSuperiorOfLimit() const;
	void FindBestSolution();
	void calculeProbabilite();
	void employedBeePhase();
	void onLookerBeePhase();
	void scoutBeePhase();
};
```

Note classe myAlgorithm nous sert principalement à définir toutes les fonctions qui nous sont utiles dans l'ABC. Notamment les trois grandes phases où nous avons besoin de trouver la fitness maximale, la meilleure solution ou encore calculer des probabilités liées aux solutions.

```
class FoodSource {
public:
	FoodSource();
	FoodSource(vector<double> coords, double objectifValue, double
fitnessValue, int trial = 0);

	double getCoordAt(int i) const;
	vector<double> getCoords() const;
	double getObjectifValue() const;
	double getFitnessValue() const;
	int getTrial() const;
	void setCoordAt(int i, double valeur);
	void setObjectifValue(double objectifValue);
	void setFitnessValue(double fitnessValue);
	void incrementeTrial();
	void resetTrial();
	void printFoodSource() const;
private:
	vector<double> d_coords;
	double d_objectifValue;
	double d_fitnessValue;
	int d_trial;
};
```

Notre classe FoodSource implémente une solution dans l'ABC.

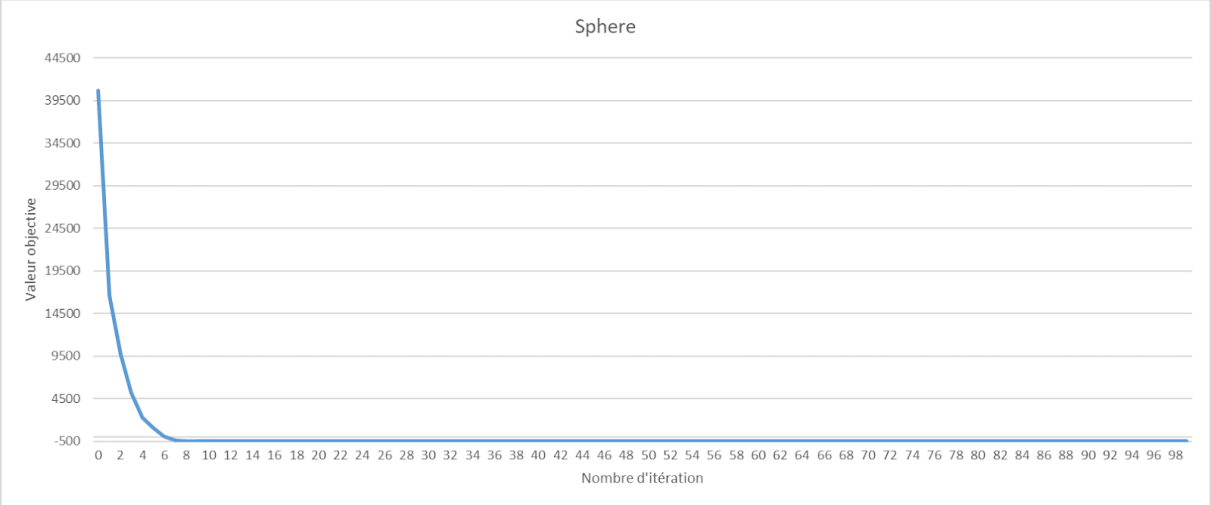
III. Résultats & benchmarks

Tous les benchmarks suivants ont été faits dans les mêmes conditions.

- Taille de la population = 100
- Nombre d'exécution = 30
- Dimension du problème = 30
- Nombre d'itération max = 5000

1. Sphere

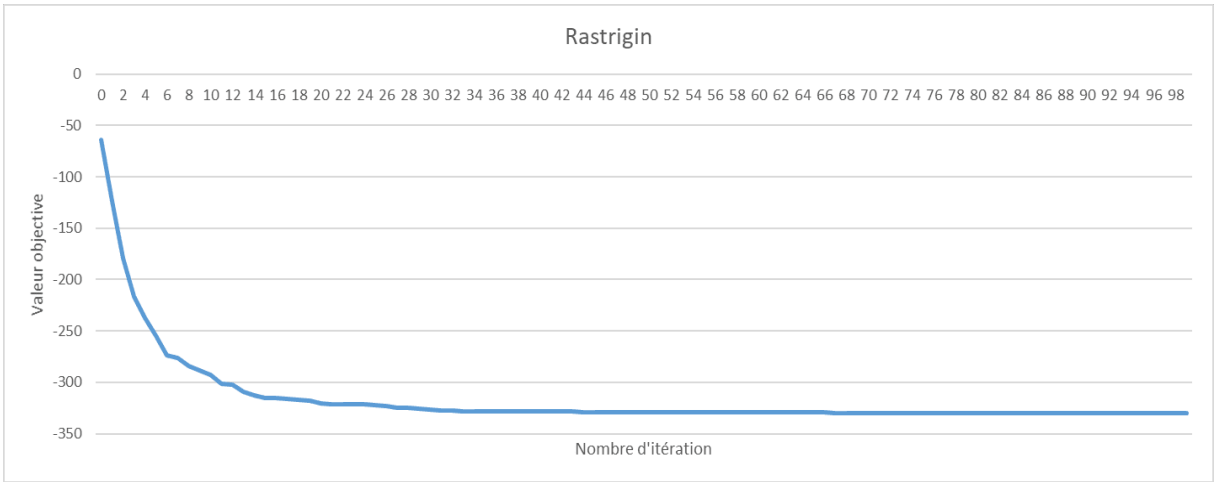
$f(X) = f_{bias} + \sum_{i=1}^{D} X_i^2$	$X = [X_1, X_2, \dots, X_n]$	$X \in [-100,100]$



On peut remarquer que notre algorithme tend vers le minimum global (le biais), ici -450. Il trouve la valeur du biais en environ 7 itérations de l'algorithme.

2. Rastrigin

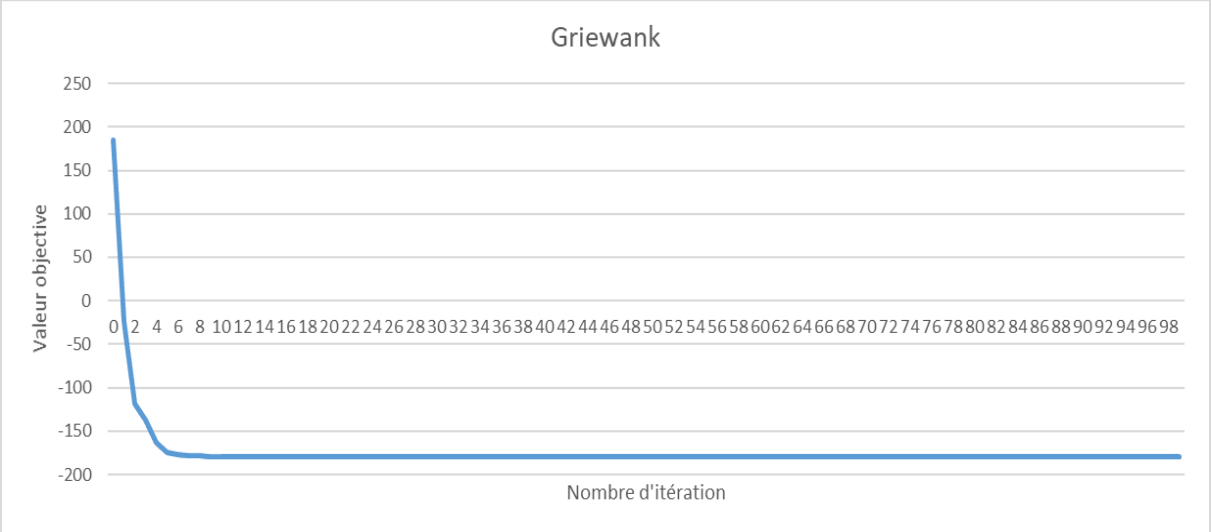
$f(X) = f_{bias} + \sum_{i=1}^{D} X_i^2 - 10\cos(2\pi * X_i) + 10)$	$X = [X_1, X_2, \dots, X_n]$	$X \in [-5,5]$



On peut remarquer que notre algorithme tend vers le minimum global (le biais), ici -330. Il trouve la valeur du biais en environ 32 itérations de l'algorithme.

3. Griewank

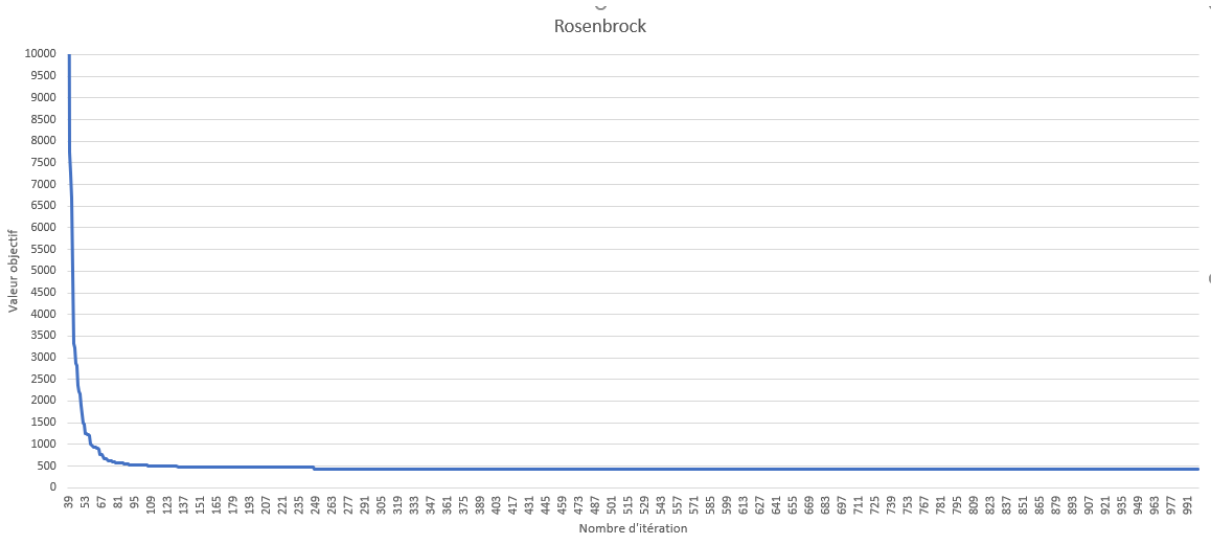
$f(X) = f_{bias} + \sum_{i=1}^{D} \frac{X_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{X_i}{\sqrt{i}}\right) + 1$	$X = [X_1, X_2, \dots, X_n]$	$X \in [-600,600]$



On peut remarquer que notre algorithme tend vers le minimum global (le biais), ici -180. Il trouve la valeur du biais en environ 7 itérations de l'algorithme.

4. Rosenbrock

$f(X) = f_{bias} + \sum_{i=1}^{D-1} \left(100\left(X_i^2 - X_{i+1}\right)^2 + (X_i - 1)^2\right)$	$X = [X_1, X_2, \dots, X_n]$	$X \in [-100,100]$



On peut remarquer que notre algorithme tend vers le minimum global (le biais), ici 390. Il ne trouve pas la valeur du biais mais s'en approche après 5000 itérations.

Comparaison

		ABC	Firefly
Sphere	Moyenne	-449.174	-449,997
	Ecart Type	0.039	0,002
Rastrigin	Moyenne	-329.375	-329,999
	Ecart Type	0.652	0,0001
Griewank	Moyenne	-179.961	-179,971
	Ecart Type	0.063	0,041
Rosenbrock	Moyenne	391.101	391,582
	Ecart Type	1.457	1,862

IV. Problèmes rencontrés

1. La documentation faible à la bibliothèque

Comme dans tous les autres projets, on est amené à faire les recherches dans toutes les sources d'information possibles.

En ayant accès à la bibliothèque universitaire, on n’a pas pu trouver les informations nécessaires pour ce projet.

2. Sources en lignes en langue étrangères

La bibliothèque n'était pas assez suffisante pour nous aider, on a dû utiliser internet.

Certains articles, et des vidéos sur youtube, sont dans la plupart des cas dans des langues étrangères (principalement anglaises).

3. Absence de membre du groupe

Le projet était à faire en groupe de quatre, mais à cause de l’absence répétée de deux de nos membres, on a dû travailler sur le projet qu’à deux. On n’a de ce faite, travaillés sur plus de tâches qu’habituellement.

Conclusion

Pour conclure, ce travail en projet a été sur un plan personnel très bénéfique, me permettant d'apprendre d'une part le fonctionnement des algorithmes génétiques et d'autre part, malgré l'absentéisme de deux de nos membres, la cohésion de groupe nécessaire à l'aboutissement de travaux de cette ampleur. Je tiens ainsi à remercier mes camarades de groupe actifs, ayant été particulièrement investis dans cette période où le contexte incertain est propice au manque de concentration.