

Prise en main du programme

Ce didacticiel explique comment exécuter et analyser une optimisation avec le programme **OptimiseurRL**. Deux problèmes d'optimisation seront utilisés pour parcourir les principales fonctionnalités du programme. Le premier est un problème mono-objectif non contraint, tandis que le second est un problème multi-objectifs contraint.

Avant de commencer, assurez-vous d'avoir installé les dépendances nécessaires spécifiées dans le fichier « requirements.txt », et lire le README.

1. Problème 1 : Optimisation mono-objectif sans contrainte

Il s'agit d'un problème non contraint à un seul objectif. Le nombre de variable pour cet exemple a été pris à 2.

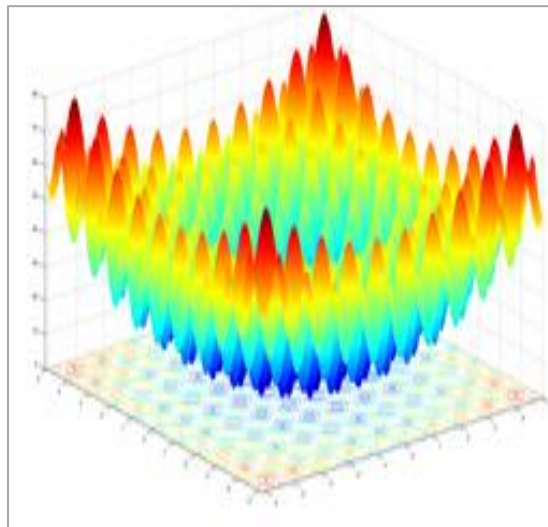
1.1. Définition du problème :

Fonction objectif : minimiser la *fonction de Rastrigin* dans l'intervalle $(-5.12 ; 5.12)$.

C'est une fonction populaire pour les tests d'optimisation en raison de ses nombreuses oscillations et de ses nombreux minima locaux. Elle est utilisée pour évaluer la capacité des algorithmes à traiter les problèmes d'optimisation multimodaux. Son expression est donnée comme suit :

$$f(\mathbf{x}) = An + \sum_{i=1}^d (x_i^2 - A \cos(2\pi x_i))$$

où A est une constante et d est le nombre de dimensions.



Le minimum global est obtenu au point $(0, 0)$, pour lequel la fonction vaut 0.

1.2. Paramétrage des données d'entrées du programme

Fournissez les données du problème dans un fichier YAML nommé deck.yaml en suivant rigoureusement les instructions consignées dans ledit fichier, tout en respectant la syntaxe. Une fois terminé, veuillez enregistrer vos modifications.

Étape 1 : Choisir le type d'exécution -- > section 2

Choisir le type d'exécution du programme en mode " entraînement ", puis renseigner les sections 3.1 et 4 du fichier YAML.

```
#----- SECTION 2 ----- CHOIX DU TYPE D'EXECUTION

# Choisir le type d'exécution du programme (" entraînement " ou " exploitation ")

#-- entraînement : executer le programme en mode entraînement revient à l'utilisation
des fonctions de test présentées
# à la section 1 comme objectifs. Si vous choisissez ce mode, vous devez
renseigner les sections 3.1 et 4.

#-- exploitation : executer le programme en mode exploitation signifie que vous
voulez résoudre un problème d'optimisation
# spécifique. Dans ce cas, vous devez renseigner les sections 3.2 et 4.

type_execution: !!str entraînement
```

Étape 2 : Formuler le problème d'optimisation --> section 3.1

A cette étape, vous devez entrer la fonction objectif à minimiser sous forme vectorisée. Par la suite, laissez le dictionnaire de contraintes vide (le problème est non contraint). Pour finir, vous devez définir le nombre de variables à considérer ainsi que leurs bornes.

```
#--- SECTION 3.1. --- ENTRAÎNEMENT DE L'ALGORITHME

# Fonction(s) objectif (type dictionnaire)
dict_objectif_training:
  rastrigin_function: |
    lambda x: 10 * len(x) + np.sum(x ** 2 - 10 * np.cos(2 * np.pi * x))

# Equations de contraintes (type dictionnaire)
dict_contrainte_training: {}

# Bornes de variables (type : liste de tuple)
n_variables_training: !!int 2
bornes_variable_training:
  - [-5.12, 5.12]
```

Étape 3 : Choix de l'algorithme d'optimisation --> section 4

Cette étape consiste à choisir l'algorithme d'apprentissage par renforcement que vous désiriez utiliser pour optimiser la fonction objectif. Vous avez le choix entre A2C, DDPG, PPO, SAC et TD3. Éventuellement, vous pouvez modifier les hyperparamètres de l'algorithme. Pour finir, vous devez définir le nombre d'itérations (nombre d'épisodes).

```
#----- SECTION 4 ----- PARAMETRES DES ALGORITHMES DE RENFORCEMENT

# Choisir l'algorithme d'apprentissage (A2C, DDPG, PPO, SAC, TD3)
#--- A2C, variant of Asynchronous Advantage Actor Critic (A3C) algorithm
#--- DDPG, Deep Deterministic Policy Gradient algorithm
#--- PPO, Proximal Policy Optimization algorithm
#--- SAC, Soft Actor Critic algorithm
#--- TD3, Twin Delayed DDPG algorithm
choix_algorithme: A2C

# Entrer le nombre d'episodes d'exécution de l'algorithme
#--- si A2C, le nombre d'episode doit être multiple de 500
#--- si DDPG, le nombre d'episode doit être multiple de 4
#--- si PPO, le nombre d'episode doit être multiple de 2048
#--- si SAC, le nombre d'episode doit être multiple de 4
#--- si TD3, le nombre d'episode doit être multiple de 4
n_episodes: !!int 20_000
```

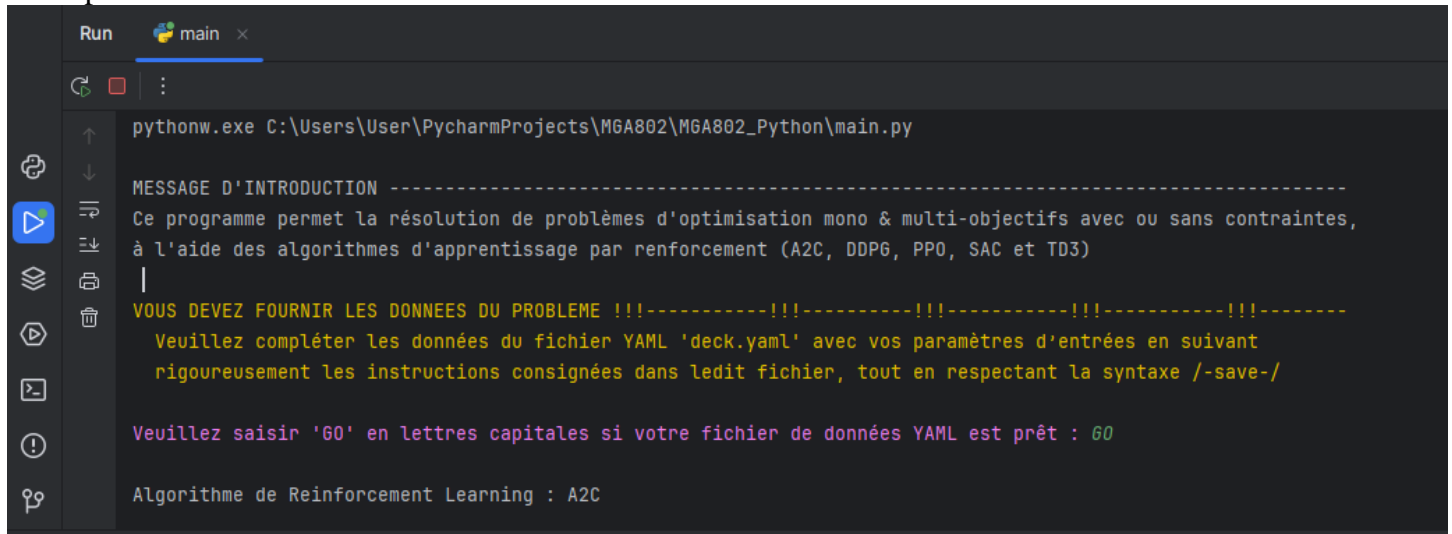
Dans cet exemple comme vous pouvez le constater, l'algorithme A2C a été choisi et le nombre d'itérations a été fixé à 20 mille.

Note : Attention au nombre d'itération trop grand, car le temps d'exécution peut être assez long !

Étape 4 : lancement du programme --> fichier *main.py*

La dernière étape consiste à exécuter le script *main.py* en utilisant la commande : *python main.py*

Suivez les instructions affichées dans le terminal et saisissez '**GO**' en lettres capitales lorsque vos données YAML sont prêtes. Après l'exécution, le programme affichera les résultats de l'optimisation ainsi que les graphiques correspondants.



```
Run main x
pythonw.exe C:\Users\User\PycharmProjects\MGA802\MGA802_Python\main.py

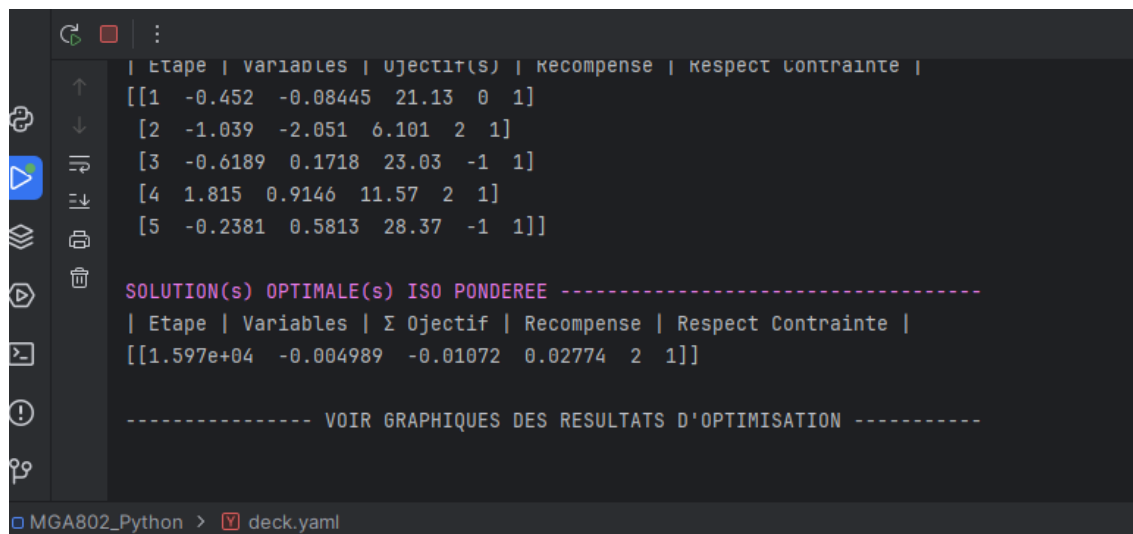
MESSAGE D'INTRODUCTION -----
Ce programme permet la résolution de problèmes d'optimisation mono & multi-objectifs avec ou sans contraintes,
à l'aide des algorithmes d'apprentissage par renforcement (A2C, DDPG, PPO, SAC et TD3)
|
VOUS DEVEZ FOURNIR LES DONNEES DU PROBLEME !!!-----!!!-----!!!-----!!!-----!!!-----!!!-----
Veuillez compléter les données du fichier YAML 'deck.yaml' avec vos paramètres d'entrées en suivant
rigoureusement les instructions consignées dans ledit fichier, tout en respectant la syntaxe /-save-/

Veuillez saisir 'GO' en lettres capitales si votre fichier de données YAML est prêt : GO

Algorithme de Reinforcement Learning : A2C
```

1.3. Résultats

Un minimum local est obtenu au point $(-0.004989, -0.01072)$, pour lequel la fonction vaut 0.02774 avec l'optimiseur RL, après 100 milles d'itérations.



```

| Etape | variables | Objectif(s) | Recompense | Respect Contrainte |
[[1 -0.452 -0.08445 21.13 0 1]
 [2 -1.039 -2.051 6.101 2 1]
 [3 -0.6189 0.1718 23.03 -1 1]
 [4 1.815 0.9146 11.57 2 1]
 [5 -0.2381 0.5813 28.37 -1 1]]

SOLUTION(s) OPTIMALE(s) ISO PONDEREE -----
| Etape | Variables | Σ Objectif | Recompense | Respect Contrainte |
[[1.597e+04 -0.004989 -0.01072 0.02774 2 1]]

----- VOIR GRAPHIQUES DES RESULTATS D'OPTIMISATION -----

MGA802_Python > deck.yaml
```

La figure 1 présente les résultats des graphiques de l'optimisation. On peut constater qu'au fil des itérations, l'algorithme continue à apprendre. Cependant, au regard de la disparité des actions (variables), on constate que le nombre d'itérations n'est pas suffisants.

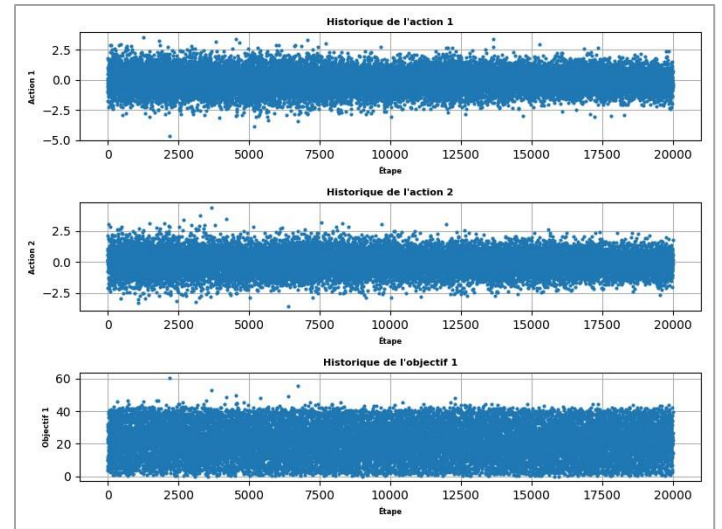
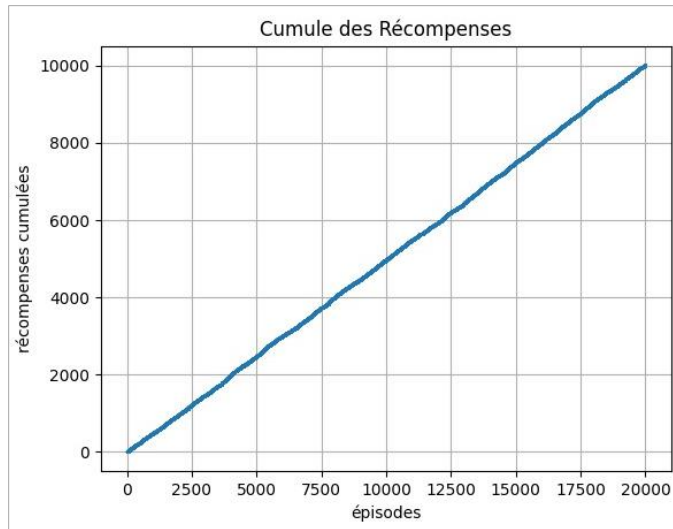


Figure 1. Courbe cumulée des récompenses (gauche) – Historique des actions et objectif (droite)

2. Problème 2 : Optimisation multi-objectifs avec contrainte

Ce problème concerne une optimisation multi-objectifs d'une poutre en I avec deux objectifs et deux contraintes.

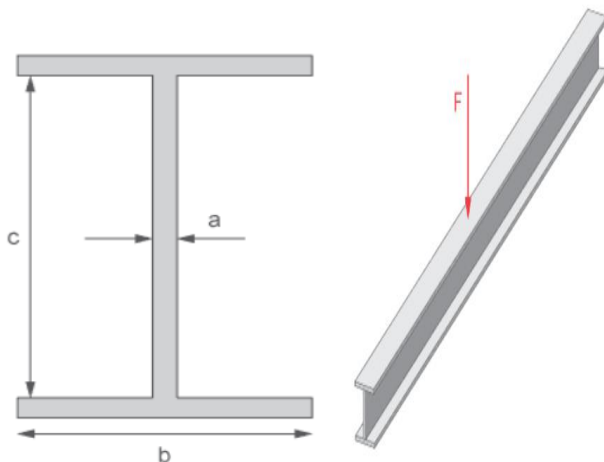
2.1. Définition du problème

Les paramètres à optimiser sont les dimensions a , b , c , c'est-à-dire *l'épaisseur de la section, la largeur de la section et la hauteur de la section*.

$$a = [1,10]\text{mm} \quad b = [50,150]\text{mm} \quad c = [50 \text{ } 250] \text{ mm}$$

$$L = 3 \text{ m} ; E = 210^{11} \text{ Pa} ; \text{densité} = 8 \text{ } 000 \text{ kg/m}^3. F = 2 \text{ } 000 \text{ N.}$$

Les extrêmes de la poutre sont considérés comme contraints par deux appuis simples (rotations autorisées).



$$I = \frac{ac^3}{12} + 2 \left[\frac{ba^3}{12} + ab \left(\frac{a}{2} + \frac{c}{2} \right)^2 \right]$$

$$\sigma = \frac{FL}{4I} \left(a + \frac{c}{2} \right)$$

$$d = \frac{FL^3}{48EI}$$

$$W = (2ab + ca)L\rho$$

Objectifs d'optimisation :

- Minimiser le poids W de la poutre
- Minimiser le déplacement maximale d

Contraintes à respecter :

- Contrainte maximale σ inférieure à 100 MPa
- Poids maximum inférieur à 20 kg

2.1. Paramétrage des données d'entrées du programme

Fournissez les données du problème dans un fichier YAML nommé `deck.yaml` en suivant rigoureusement les instructions consignées dans ledit fichier, tout en respectant la syntaxe. Une fois terminé, veuillez enregistrer vos modifications.

Étape 1 : Choisir le type d'exécution --> section 2

Choisir le type d'exécution du programme en mode " exploitation ", puis renseigner les sections 3.2 et 4 du fichier YAML.

```
#----- SECTION 2 ----- CHOIX DU TYPE D'EXECUTION

# Choisir le type d'exécution du programme (" entraînement " ou " exploitation ")

#-- entraînement : executer le programme en mode entraînement revient à l'utilisation
des fonctions de test présentées
# à la section 1 comme objectifs. Si vous choisissez ce mode, vous devez
renseigner les sections 3.1 et 4.

#-- exploitation : executer le programme en mode exploitation signifie que vous
voulez résoudre un problème d'optimisation
# spécifique. Dans ce cas, vous devez renseigner les sections 3.2 et 4.

type_execution: !!str exploitation
```

Étape 2 : Formuler le problème d'optimisation --> section 3.2

A cette étape, vous devez entrer les fonctions objectif à minimiser sous forme vectorisée, ainsi que les expressions des équations de contrainte. Pour finir, vous devez définir les bornes de variables.

```
#--- SECTION 3.2. --- EXPLOITATION DE L'ALGORITHME

# Fonction(s) objectif (type dictionnaire)
dict_objectif_exploitation:
  def1: |
    lambda x: (((2000) * (3)**3 / (48 * (2 * 10**11) * (x[0] * (x[2]**3) / 12 + 2 * (x[1]
* (x[0]**3) / 12 +
    ((x[2] / 2 + x[0] / 2)**2) * x[0] * x[1])))))*10**0
  weight: |
    lambda x: (2 * x[0] * x[1] + x[2] * x[0]) * (3) * (8000)
  weight2: |
    lambda x: (2 * x[1] * x[1] + x[2] * x[0]) * (3) * (8000)

# Equations de contraintes (type dictionnaire)
dict_contrainte_exploitation:
  stress_max: |
    lambda x: 0.25 * (2000) * (3) * (x[2] / 2 + x[0]) / (x[0] * (x[2]**3) / 12 + 2 *
(x[1] * (x[0]**3) / 12 +
    ((x[2] / 2 + x[0] / 2)**2) * x[0] * x[1])) <= 100*10**6
  weight_max: |
    lambda x: (2 * x[0] * x[1] + x[2] * x[0]) * (3) * (8000) <= 20

# Bornes de variables (type : liste de tuple)
bornes_variable_exploitation:
  - [0.001, 0.01]
  - [0.05, 0.15]
  - [0.05, 0.25]
```

Étape 3 : Choix de l'algorithme d'optimisation --> section 4

L'algorithme DDPG a été choisi pour la résolution du problème, pour 1000 itérations.

```
#----- SECTION 4 ----- PARAMETRES DES ALGORITHMES DE RENFORCEMENT

# Choisir l'algorithmme d'apprentissage (A2C, DDPG, PPO, SAC, TD3)
#--- A2C, variant of Asynchronous Advantage Actor Critic (A3C) algorithm
#--- DDPG, Deep Deterministic Policy Gradient algorithm
#--- PPO, Proximal Policy Optimization algorithm
#--- SAC, Soft Actor Critic algorithm
#--- TD3, Twin Delayed DDPG algorithm
choix_algorithmme: DDPG

# Entrer le nombre d'episodes d'exécution de l'algorithmme
#--- si A2C, le nombre d'episode doit être multiple de 500
#--- si DDPG, le nombre d'episode doit être multiple de 4
#--- si PPO, le nombre d'episode doit être multiple de 2048
#--- si SAC, le nombre d'episode doit être multiple de 4
#--- si TD3, le nombre d'episode doit être multiple de 4
n_episodes: !!int 2_000
```

Étape 4 : lancement du programme --> fichier *main.py*

La dernière étape consiste à exécuter le script *main.py* en utilisant la commande : `python main.py`

Suivez les instructions affichées dans le terminal et saisissez '**GO**' en lettres capitales lorsque vos données YAML sont prêtes. Après l'exécution, le programme affichera les résultats de l'optimisation ainsi que les graphiques correspondants.

```
MESSAGE D'INTRODUCTION -----
Ce programme permet la résolution de problèmes d'optimisation mono & multi-objectifs avec ou sans contraintes,
à l'aide des algorithmes d'apprentissage par renforcement (A2C, DDPG, PPO, SAC et TD3)

VOUS DEVEZ FOURNIR LES DONNEES DU PROBLEME !!!-----!!!-----!!!-----!!!-----!!!-----
Veuillez compléter les données du fichier YAML 'deck.yaml' avec vos paramètres d'entrées en suivant
rigoureusement les instructions consignées dans ledit fichier, tout en respectant la syntaxe /-save-/

Veuillez saisir 'GO' en lettres capitales si votre fichier de données YAML est prêt : GO

Algorithme de Reinforcement Learning : DDPG

MGA802_Python > main.py
```

2.2. Résultats

Pour ce problème de poutre, le cumule de récompenses de l'agent ne devient positif qu'à partir de la 750^{ème} itération.

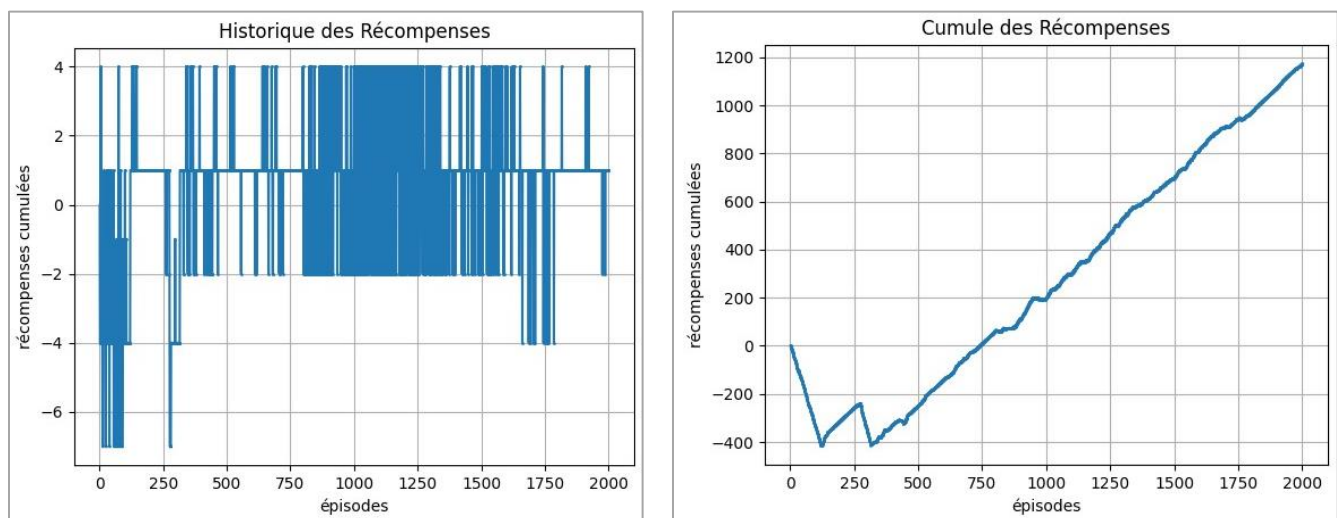


Figure 2. Historique des récompenses (gauche) – Courbe cumulée des récompenses (droite)

La figure 3 présente les historiques des valeurs des variables (action) et des objectifs. Les actions 1, 2 et 3 correspondent aux variables de design a , b et c respectivement.

Les objectif 1 et 2 quant à eux correspondent aux objectifs de déflexion d et poids W respectivement.

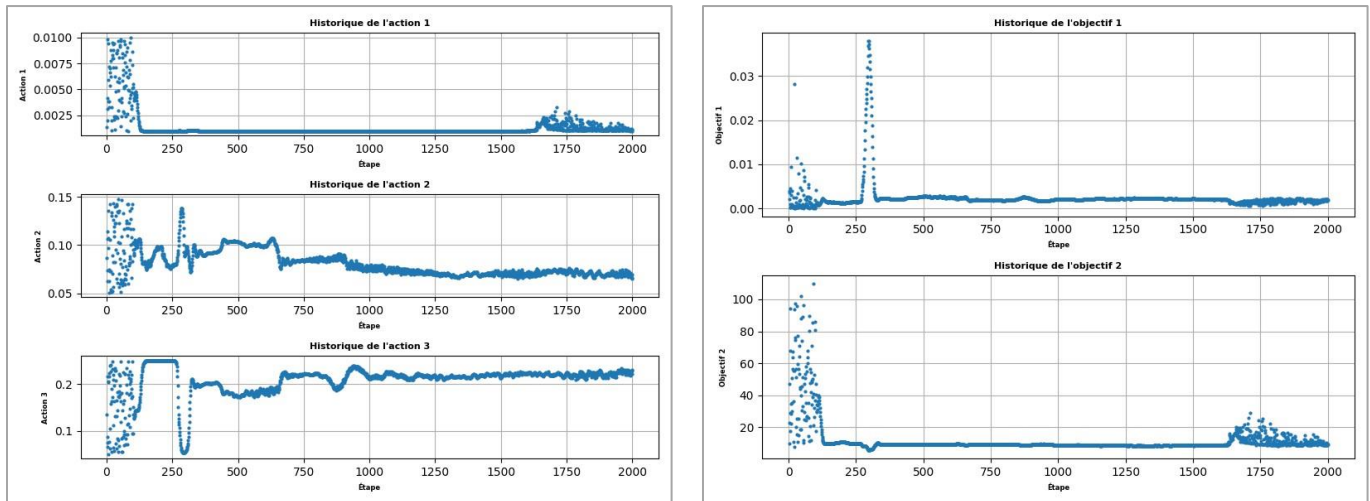


Figure 3. Historique des actions (gauche) – Historique des objectifs (droite)

La figure 4 présente l'allure du front de Pareto du problème d'optimisation. La solution optimale n'est pas unique comme dans un cas d'optimisation à un seul objectif, mais plutôt un ensemble de conceptions optimales se situant sur le front de Pareto.

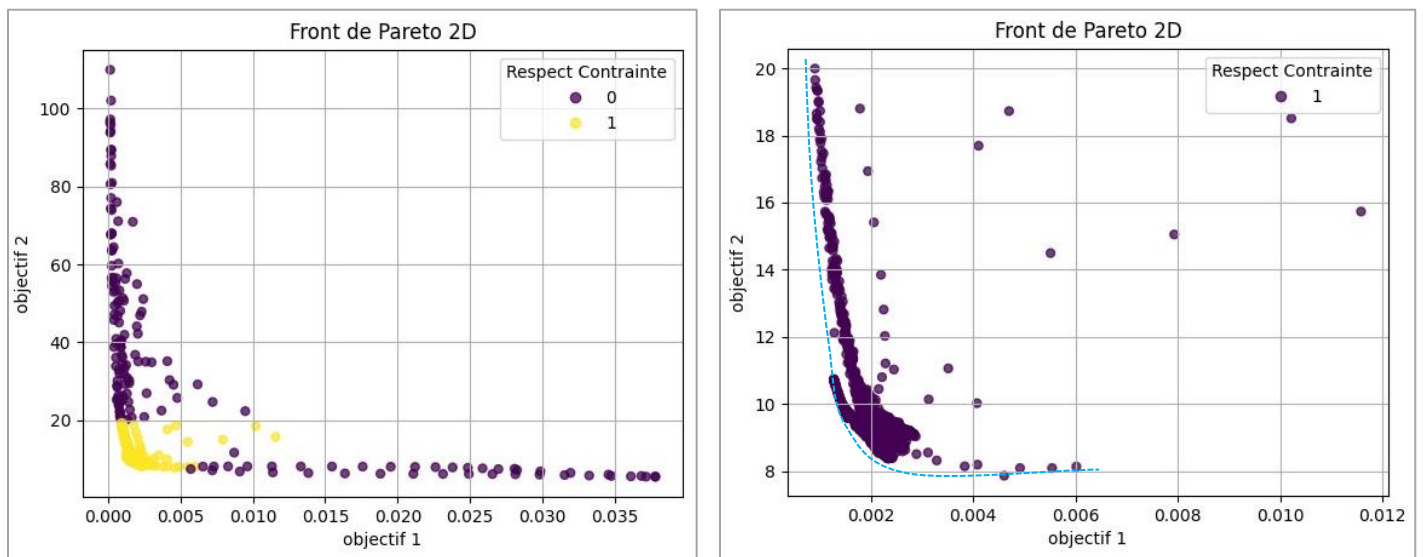


Figure 4. Front de Pareto 2D solutions brute (gauche) – Front de Pareto 2D solutions faisables (droite)