Méthode d'optimisation par Essaims de particules

Projet de fin de semestre 2, MAM3

Dougnac Jade & Barnetche Carine & Bricout Guillaume & Dubes Hugo

Polytech Nice Sophia

29 mai 2020

Sommaire

Sommaire

Introduction et motivations

Introduction

Notre projet consiste à étudier l'optimisation par essaim particulaire (PSO). L'optimisation est un moyen mathématique permettant de modéliser et résoudre des problèmes numériques en minimisant ou maximisant des fonctions sur un ensemble. Elle est omniprésente dans tous les domaines et ne cesse d'évoluer depuis Euclide. La méthode PSO va permettre de faire converger une fonction vers un ou plusieurs points que l'on appelle optimums. Ce projet sera codé sous Python.

Introduction et motivations

Motivations

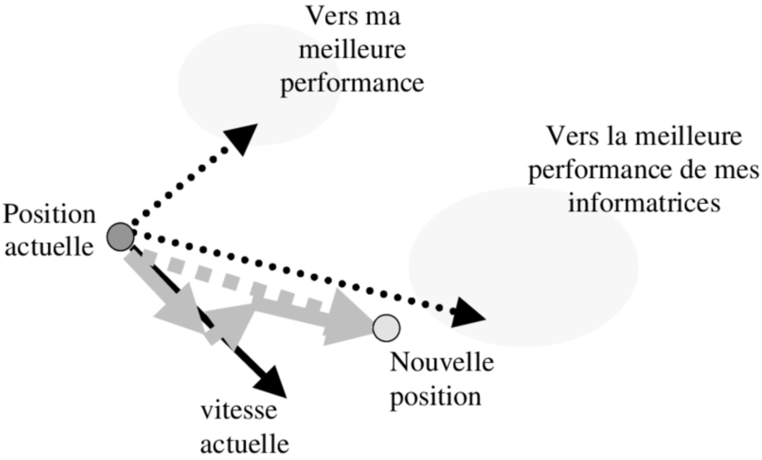
Pour la résolution de nombreux problèmes, il est difficile de trouver une solution déterministe en un temps raisonnable. C'est donc pour cela que nous faisons appel à des méthodes dites métaheuristiques. Une métaheuristique est un algorithme d'optimisation dont le but est de résoudre un problème difficile. Cet algorithme va tendre vers un optimum global de la fonction etudiée. Ces méthodes sont souvent utilisées car elles sont généralistes et sont adaptables pour tout type de problèmes d'optimisation.

Introduction et motivations

Élements de la PSO

Pour appliquer la PSO il faut définir un espace de recherche constitué de particules et une fonction objectif à optimiser. Le principe de l'algorithme est de déplacer ces particules afin qu'elles trouvent *l'optimum*.

Introduction et motivations



Sommaire

Algorithme

Fonction f

- X est un vecteur à 2 dimensions (x_1, x_2)

$$\begin{aligned} f : \mathbb{R}^2 &\rightarrow \mathbb{R} \\ (x_1, x_2) &\mapsto f(x) \end{aligned}$$

Notre essaim contient N particules d'où X^i avec $0 < i \leq N$ et on regarde le déplacement des points pour un temps k donné avec $0 < k \leq K_{max}$

Algorithme

Composantes qui définissent X_k

- V_k qui est la vitesse au point k
- $pbestx^i$ qui est la meilleure position de la particule
- $pbest = f(pbest_X)$ qui est le "personal best" de la particule
- $best_X$ qui représente la meilleure position de toutes les particules
- $gbest = f(best_X)$ qui représente le minimum global

Calcul de la vitesse V$_{k+1}$

Présence des constantes ω, ϕ_1, ϕ_2

Présence de nombres aléatoires entre 0 et 1 U_1 et U_2

$$V_{k+1}^i = V_k^i + \phi_1 * U_1 * (pbest_X - X_k^i) + \phi_2 * U_2 * (best_X - X_k^i)$$

Algorithme

Calcul de la valeur X^i_{k+1}

$X^i_{k+1} = X^i_k + V^i_k$

On a :

X^i_k : position de la ième particule au temps k

On génère X_0 et V_0 au début de notre programme avec $V_0 = 0$ pour toutes les particules

Sommaire

Test de validation

Ces tests de validation nous permettent de vérifier que notre code fonctionne bien avec des fonctions quelconques. Parmi les tests de validation, nous avons choisi :

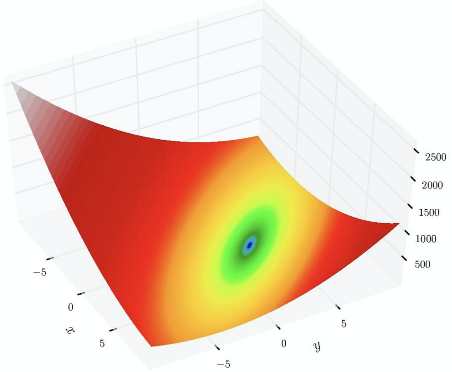
- La fonction Booth
- La fonction Rastrigin

La fonction booth se définit ainsi :

$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$

Son domaine de recherche est : $-10 \leq x_i \leq 10$

Son minimum global est : $f(1,3) = 0$



Tests de validation

Fonction booth

Paramètres

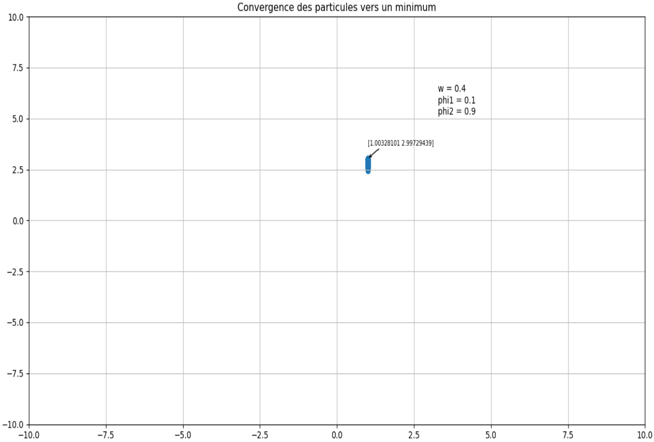
On choisit $\omega = 0.4$, $\phi_1 = 0.1$ et $\phi_2 = 0.9$. On se place dans l'intervalle]-10; +10[
On prend 200 nombres de particules et le nombre d'itérations maximales Nmax=100.

Observations

Nous remarquons que notre erreur est proche de 0 ce qui signifie que l'on approche bien du minimum. De plus, toutes les particules convergent bien vers le minimum.

Tests de validation

Fonction booth



Tests de validation

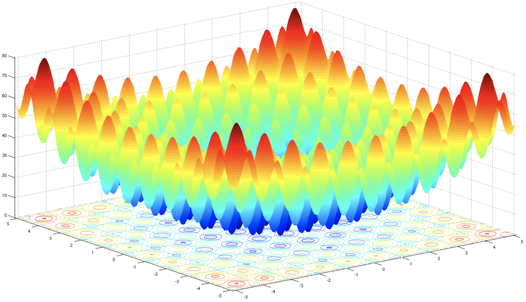
Fonction rastrigin

Nous avons étudié en deuxième fonction test la fonction rastrigin :

f(x)=$An + \sum_{i=1}^{n} x_i^2 - Acos(2\pi * x_i)$

Son domaine de recherche est : -5.12 $\leq x_i \leq$ 5.12

Son minimum global est : f(0,...,0) $=$ 0



Paramètres

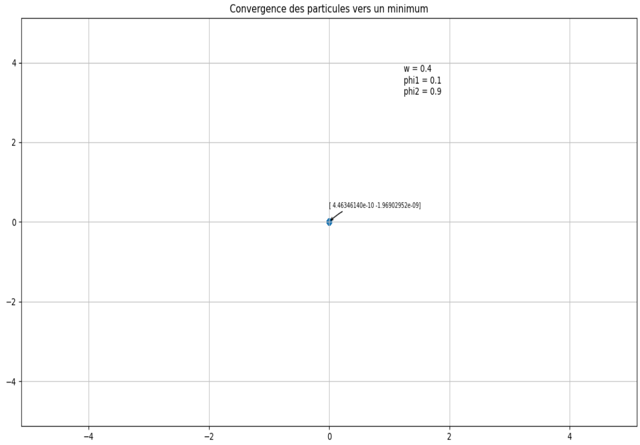
On choisit $\omega = 0.4$, $\phi_1 = 0.1$ et $\phi_2 = 0.9$. On se place dans l'intervalle]-5.12; +5.12[
On prend 200 nombres de particules et le nombre d'itérations maximales Nmax=100.

Observations

Nous remarquons que de même, l'erreur est proche de 0 et toutes les particules convergents bien vers le minimum.

Tests de validation

Fonction rastrigin



Sommaire

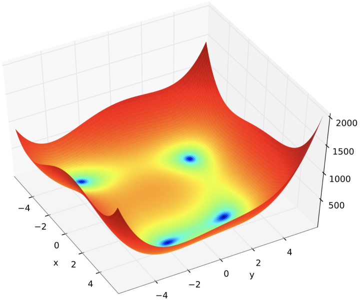
Etude paramétrique himmelblau

Influence de la taille de l'essaim

Nous avons eu à étudier la fonction Himmelblau qui se définit ainsi :

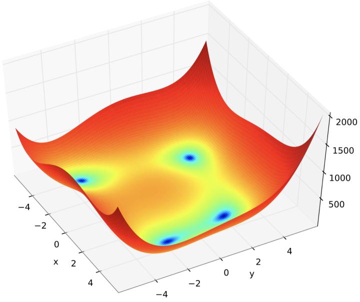
$f(x,y)=(x^2+y-11)^2+(x+y^2-7)^2$

Son domaine de recherche est : $-5 \leq x_i \leq 5$



Ses minimums glogaux sont :

$$\text{Min} = \begin{cases} f(3.0, 2.0) = 0.0 \\ f(-2.805118, 3.131312) = 0.0 \\ f(-3.779310, -3.283186) = 0.0 \\ f(3.584428, -1.848126) = 0.0 \end{cases}$$



Etude paramétrique himmelblau

Influence de la taille de l'essaim


```
Taille de l'essaim : 5
Erreur = [0.5504019058187315, 0.012993849576906857]
```

```
Taille de l'essaim : 50
Erreur = [0.020526870826029233, 0.0016952323161176786]
```

```
Taille de l'essaim : 500
Erreur = [0.00018530695694352062, 6.842553463970447e-05]
```

Analyse des valeurs

En prenant 100 itérations, on remarque que la distance entre le minimum de l'essaim et le minimum théorique réduit fortement (avec un facteur > 10) et que lorsque l'on prend une taille de l'essaim de 500, on a une erreur qui se rapproche de 10^{-6} pour les y ce qui nous montre donc que la taille de l'essaim possède une influence importante sur la convergence de la fonction. En effet, au plus elle augmente, au plus on obtient un résultat précis.

Etude paramétrique himmelblau

Influence de paramètres PSO

Introduction

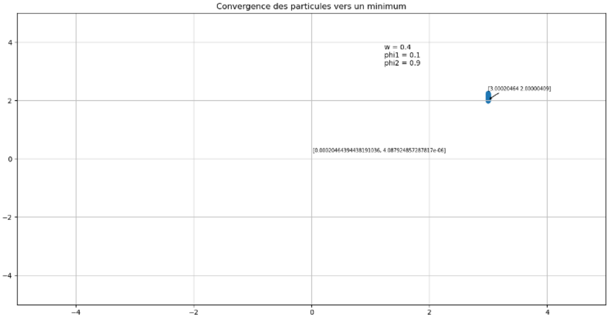
Pour une meilleure compréhension nous avons fait l'étude en regardant autour du minimum atteint en (3,2). Nous obtenons des résultats similaires pour les autres minimums.

Paramètres fixes

Nous avons dans un premier temps fixés les paramètres $\omega = 0.4$, $\phi_1 = 0.1$ et $\phi_2 = 0.9$.
Avec ces paramètres tous les points convergents vers un minimum de la fonction et se retrouvent sur le minimum à la dernière itération. On remarque qu'on obtient la convergence de tous les points sur le minimum pour environ 25 itérations, ce qui nous paraît être le plus optimisé que l'on puisse faire.

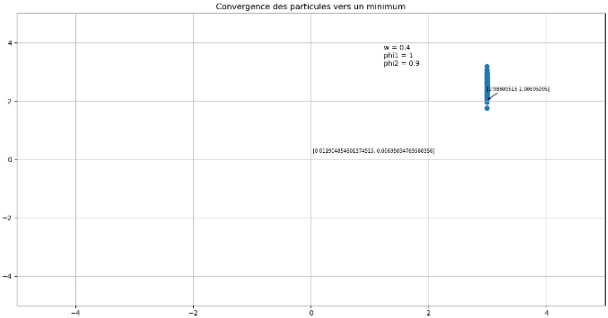
Résultat obtenu

Voici ce que l'on obtient (sur chaque graphique on a affiché l'erreur entre le minimum de l'essaim obtenu et le minimum théorique).



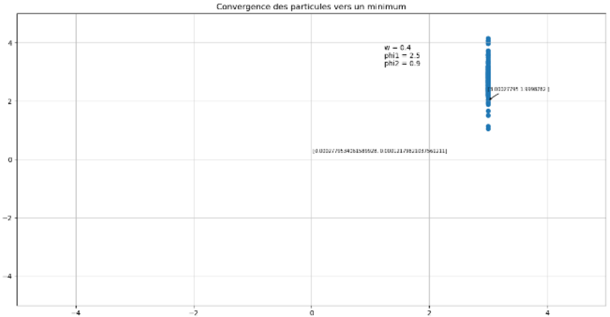
Variation de ϕ_1

Après avoir fixé $\omega = 0.4$ et $\phi_2 = 0.9$, nous avons fait varier phi1 en le faisant augmenter à chaque fois et on observe le résultat à la fin de la dernière itération. Voici ce qu'on obtient pour ϕ_1=1.



Résultat obtenu

Voici ce qu'on obtient pour ϕ_1=2.5



Méthode d'optimisation par Essaims de particules

Dougnac Jade & Barnetche Carine & Bricout Guillaume & Dubes Hugo

Introduction et motivations

Algorithme

Tests de validation

Fonction booth

Fonction rastrigin

Etude paramétrique himmelblau

Influence de la taille de l'essaim

Influence de paramètres PSO

Conclusion

Observations

Pour ϕ=2.5, 25 itérations ne suffisent pas à obtenir la convergence de tous les points, nous avons observé qu'il nous fallait fixer au moins 150 itérations maximales afin d'avoir une convergence moins dispersée. De même pour $\phi_1 = 3$ où nous avons du encore augmenter les itérations. On remarque qu'au fur et à mesure que nous augmentons ϕ_1, les points sont de plus en plus dispersés autour d'un minimum. Ainsi en augmentant ϕ_1 on augmente l'influence du « personal best » sur le chemin parcouru par les particules, ce qui fait qu'elles se dispersent un peu plus.

Observations

Cependant, quelle que soit la valeur de ϕ_1, nous arrivons toujours à afficher le minimum exact (le $best_x$), on a donc toujours convergence des particules de manière plus ou moins éparpillées vers le minimum.
Selon nos tests, avec l'augmentation de ϕ_1 il est nécessaire d'avoir un plus grand nombre d'itérations, ainsi on remarque que l'augmentation de ϕ_1 entraîne un programme moins optimisé.

Sommaire

Conclusion

L'optimisation par essaim de particules est une nouvelle méta-heuristique très efficace car, à partir de peu de modifications, elle permet la résolution d'un grand nombres de problèmes. De plus, elle fournit de très bons résultats pour des applications d'optimisation discrète alors qu'elle est définie à la base pour des problèmes d'optimisation continue.

Sommaire

Annexes

```python
import matplotlib
matplotlib.use('TkAgg')
import numpy as np
import random
import matplotlib.animation as animation
from matplotlib import pyplot as plt

#Fonction de HIMMELBLAU
def f(x, y):
    return (x **2+ y - 11)**2 + (x + y**2 - 7)**2

"""
#FONCTIONS UTILISEES POUR LE TEST DE VALIDATION
#Fonction de RASTRIGIN
def f(x,y):
    return 20 + x * x + y * y - 10 * (np.cos(2 * np.pi * x) + np.cos(2 * np.pi * y))

#Fonction de BOOTH
def f(x,y):
    return (x+2*y-7)**2+(2*x+y-5)**2
"""

Nmax = 100
nbparticules = 200
ndim = 2
phi1 = 0.1
phi2 = 0.9
w=0.4
```

Annexes

```python
def PSO():
    """
    INITIALISATION
    """
    #X et V sont des tableaux à trois dimensions, vides dans un premier temps
    #X[i] contient la position des points (X[i] = (x[i],y[i]) au temps i
    #V[i] contient la vitesse au temps i
    X = np.empty((Nmax + 1, nbparticules, ndim))
    V = np.empty((Nmax + 1, nbparticules, ndim))

    #le premier élement de X contiendra nparticules points (x,y) pris au hasard compris entre -5 et 5
    X[0] = np.random.uniform(-5, 5, (nbparticules, 2))
    #le premier élement de V est un tableau contentant des zeros de dimension nbparticules x 2
    V[0] = np.zeros((nbparticules, 2))

    #pbest_X = la meilleure position de la particule
    #pbest = f(pbest_X), c'est le "personnal best"
    #On les initialise avec la position de départ
    pbest = f(X[0][0][0], X[0][0][1])
    pbest_X = np.array(X[0][0][0], X[0][1][1])

    #best_X est la meilleure position de toutes les particules
    #gbest = f(best_X), c'est le "global best"
    best_X = np.array([X[0][0][0], X[0][0][1]])
    gbest = f(X[0][0][0], X[0][0][1])
```

Annexes

```
    ALGORITHME
    """
    for k in range(0,Nmax):
        for i in range(0, nbparticules):
            U1 = random.uniform(0, 1)
            U2 = random.uniform(0, 1)
            newV = w*V[k][i] + phi1 * U1 * (pbest_X - X[k][i]) + phi2 * U2 * (best_X - X[k][i])
            newX = X[k][i] + newV
            V[k + 1][i] = newV
            X[k + 1][i] = newX
            if (f(newX[0], newX[1]) < gbest):
                gbest = f(newX[0], newX[1])
                best_X = newX
                pbest = f(newX[0], newX[1])
                pbest_X = np.array(newX[0], newX[1])
    print('Minimum = ', best_X)
    return X, best_X

X,best_X = PSO()

#On calcule l'erreur qui est la différence entre le minimum théorique
# et le minimum de que l'on a obtenu (position du "global best")
def erreur(best_X):
    Erreur = []
    if (best_X[0] < 0 and best_X[1] > 0):
        Erreur.append(abs(-2.805118 - best_X[0]))
        Erreur.append(abs(3.131312 - best_X[1]))
    if (best_X[0] < 0 and best_X[1] < 0):
        Erreur.append(abs(-3.779310 - best_X[0]))
        Erreur.append(abs(-3.283186 - best_X[1]))
    if (best_X[0] > 0 and best_X[1] < 0):
        Erreur.append(abs(3.584428 - best_X[0]))
        Erreur.append(abs(-1.848126 - best_X[1]))
    if (best_X[0] > 0 and best_X[1] > 0):
        Erreur.append(abs(3.0 - best_X[0]))
        Erreur.append(abs(2.0 - best_X[1]))
    return Erreur
erreur=erreur(best_X)
```

Annexes

```python
"""
MODELISATION
On modélise la convergence des particules avec un graphique animé
"""
fig = plt.figure()
ax = fig.add_subplot(111)
ax.grid(True, linestyle='-', color='0.75')
ax.set_xlim([-5, 5])
ax.set_ylim([-5, 5])
scat = plt.scatter(X[0][:, 0], X[0][:, 1])
Y, Z = np.meshgrid([-5, 5], [-5, 5])

def anime(i):
    scat.set_offsets(X[i])
    return scat,

anim = animation.FuncAnimation(fig, anime, frames=np.arange(0, Nmax, 1),
                              interval=100,repeat=False)

plt.title("Convergence des particules vers un minimum")

plt.text(1.2,3.2," w = 0.4\n phi1 = 0.1\n phi2 = 0.9 ")

plt.annotate(best_X,
         xy=(best_X[0],best_X[1]), xycoords='data',
         xytext=(+0, +15), textcoords='offset points', fontsize=7,
         arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.annotate(erreur,
         xy=(erreur[0],erreur[1]), xycoords='data',
         xytext=(+2, +10), textcoords='offset points', fontsize=7)

plt.show()
```