Méthode PSO d'optimisation par Essaims de particules

Julien Choukroun & Samy David & Jessica Gourdon & Luc Sagnes

Polytech Nice Sophia

29 Mai 2020

Sommaire

Introduction

Présentation

Ce projet de fin de semestre consiste à étudier une métaheuristique : l'optimisation par essaim particulaire (PSO).

Introduction

Présentation

Ce projet de fin de semestre consiste à étudier une métaheuristique : l'optimisation par essaim particulaire (PSO).

Objectif

L'objectif de cette méthode est de trouver l'optimum d'une fonction, c'est-à-dire le point vers lequel tous les éléments vont converger.

Optimisation

Problème d'optimisation

L'optimisation en mathématiques est la sélection d'un élément « meilleur » au vu de certains critères parmi un ensemble de solutions disponibles.
Un problème d'optimisation consiste donc à maximiser ou minimiser une fonction réelle en choisissant en entrée des valeurs à partir d'un ensemble autorisé et en calculant la valeur de la fonction correspondante. Il faut noter que les problèmes de recherche de maximum ou de minimum sont symétriques. En effet, rechercher le minima de f revient à rechercher le maxima de -f.

Métaheuristique

Définition

Une métaheuristique est un algorithme d'optimisation dont le but est de résoudre un problème d'optimisation difficile que l'on ne peut pas résoudre par le biais de méthodes d'optimisation classiques.
Il existe de nombreux problèmes ne possédant pas de solution donnant un résultat en un temps raisonnable.
Dans ce cas, nous pouvons avoir recours à des méthodes heuristiques. Il s'agit de méthodes permettant d'obtenir une solution approchée, la meilleure possible, dans un délai de temps raisonnable.
Parmi celles-ci, il existe des algorithmes capables de s'adapter à une large gamme de problèmes d'optimisation différents. Il s'agit des métaheuristiques.

Principe de la PSO

Présentation

La PSO s'inspire du monde vivant. Elle se base sur la collaboration des individus entre eux pour converger progressivement vers un minimum global. Les individus seront appelés « particules ».

Principe de la PSO

Présentation

La PSO s'inspire du monde vivant. Elle se base sur la collaboration des individus entre eux pour converger progressivement vers un minimum global. Les individus seront appelés « particules ».

Objectif

Pour appliquer cette méthode, on doit définir un espace de recherche constitué de particules et une fonction à optimiser. Notre objectif est de déplacer ces particules afin qu'elles se retrouvent toutes très proches de l'optimum.

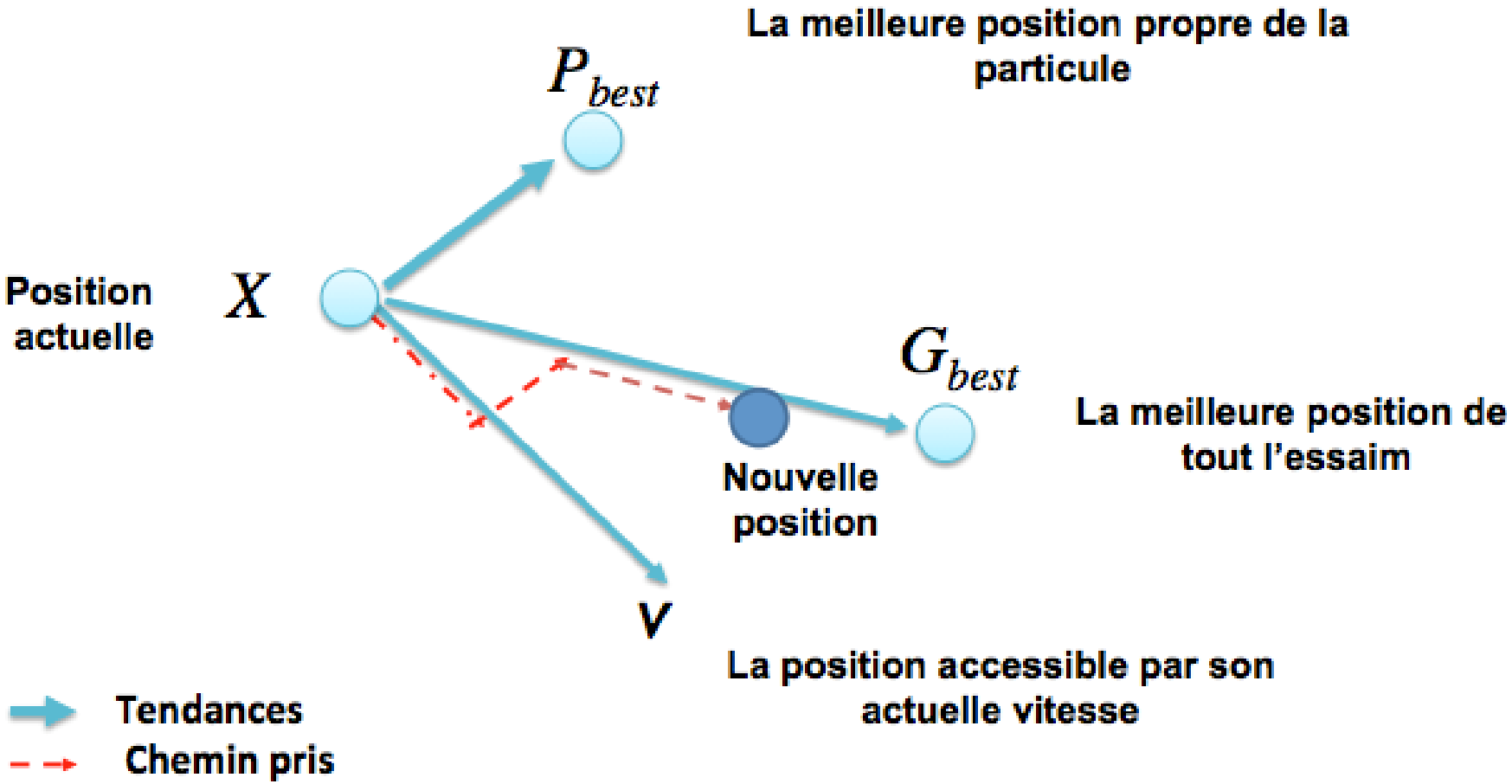
Principe de la PSO

Fonctionnement

Dans l'optimisation par essaim de particules, une équation permet de guider les particules. Le déplacement de celles-ci est influencé par trois composantes :

- L'inertie : la particule tend à suivre sa direction courante de déplacement
- L'influence personnelle : la particule tend à se diriger vers la meilleure position par laquelle elle est déjà passée
- L'influence sociale : la particule tend à se diriger vers la meilleure position atteinte par ses voisines

Principe de la PSO



Algorithme

Une particule i de l'essaim dans un espace de dimension D est caractérisée, à l'instant t, par :

- x : sa position dans l'espace de recherche
- V : sa vitesse
- xLB : la position de la meilleure solution par laquelle elle est passée, c'est le local best
- xGB : la position de la meilleure solution connue de tout l'essaim, c'est le global best
- f(xLB) : la valeur de sa meilleure solution
- f(xGB) : la valeur de la meilleure solution connue de tout l'essaim

Le déplacement de la particule i entre les itérations t et t+1 se fait selon les deux équations (1) et (2) suivantes

Algorithme

Calcul de la vitesse

$$V_i^{t+1} = wV_i^t + \phi_1 U_1^t(xLB_i^t - x_i^t) + \phi_2 U_2^t(xGB_i^t - x_i^t) \quad (1)$$

Algorithme

Calcul de la vitesse

$$V_i^{t+1} = wV_i^t + \phi_1 U_1^t(xLB_i^t - x_i^t) + \phi_2 U_2^t(xGB_i^t - x_i^t) \quad (1)$$

Calcul de la position

$$x_i^{t+1} = x_i^t + V_i^{t+1} \quad (2)$$

Algorithme

Calcul de la vitesse

$$V_i^{t+1} = wV_i^t + \phi_1 U_1^t(xLB_i^t - x_i^t) + \phi_2 U_2^t(xGB_i^t - x_i^t) \quad (1)$$

Calcul de la position

$$x_i^{t+1} = x_i^t + V_i^{t+1} \quad (2)$$

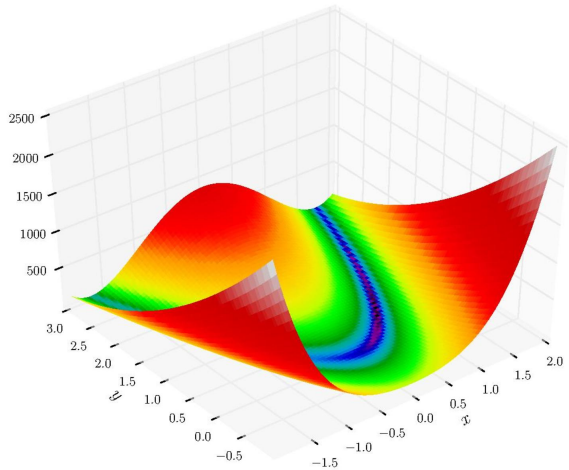
w est la vitesse d'inertie.
ϕ_1 et ϕ_2 sont des coefficients d'influence locale et globale.
U_1 et U_2 sont des constantes aléatoires tirées dans l'intervalle [0 ;1] pour éviter une agglutination autour des global best.

Algorithme

Result: Afficher la meilleure solution trouvée xGB
Initialisation des paramètres et de la taille N de l'essaim
Initialisation des vitesses et des positions aléatoires des particules dans chaque dimension de l'espace de recherche
Pour chaque particule, $xLB \leftarrow x$
Calculer f(xLB) pour chaque particule
Calculer xGB *//le meilleur xLB*
while *Condition d'arret n'est pas vérifiée* **do**
    for *i allant de 2 à N* **do**
        Calculer la nouvelle vitesse à l'aide de l'équation (1)
        Calculer la nouvelle position à l'aide de l'équation (2)
        Calculer f(xLB) pour chaque particule
        if *f(x) est meilleure que f(xLB)* **then**
            $xLB \leftarrow x$
        end
        if *f(xLB) est meilleure que f(xGB)* **then**
            $xGB \leftarrow xLB$
        end
    end
end

Test de validation

Nous avons étudier la fonction de Rosenbrock.
Cette fonction se définit ainsi :
$f(x) = \sum_{i=1}^{n-1}(100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2)$
Son domaine de recherche est $-\infty \leq x_i \leq +\infty$ avec $1 \leq i \leq n$
Son minimum global est : $f(1,1) = 0$ pour $n = 2$, $f(1,1,1) = 0$ pour $n = 3$ et $f(\underbrace{1,...,1}_{nfois}) = 0$ pour $n > 3$



Test de validation

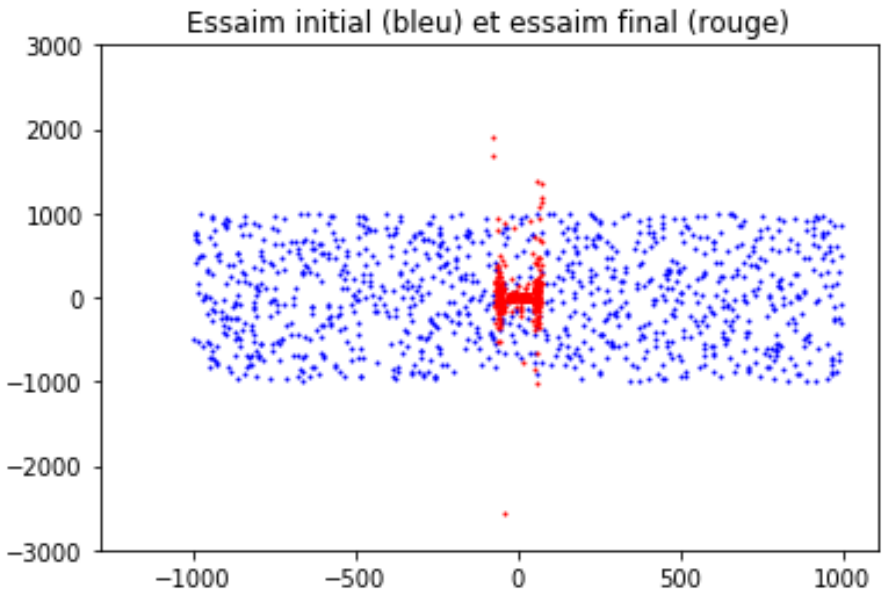
Paramètres

On choisit $w = 1$, $\phi_1 = 1$ et $\phi_2 = 1$.
On se place dans l'intervalle [-1000 ;1000].
On prend une population de 2000 et on étudie 1000 points, avec un nombre d'itérations maximales de 1000.

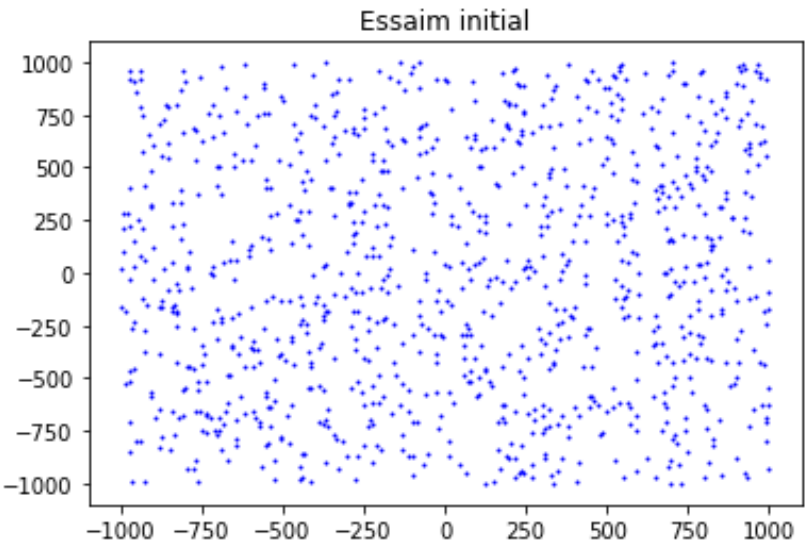
Test de validation

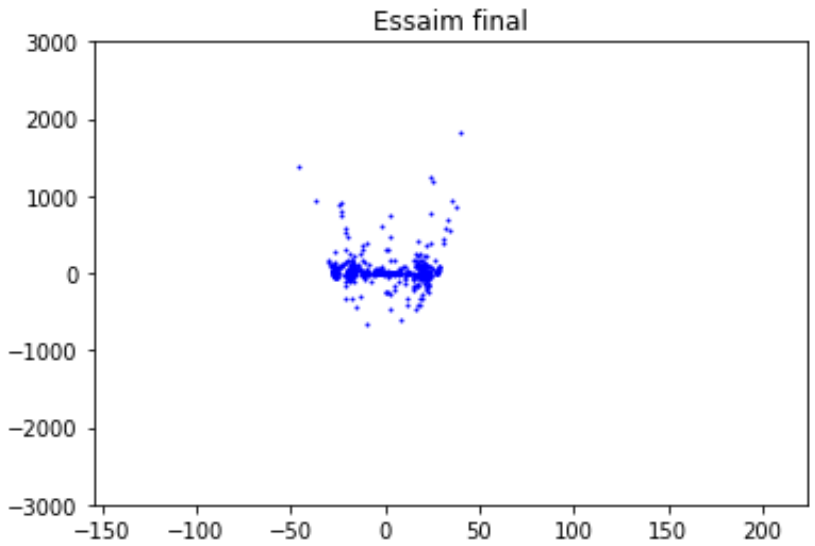
Paramètres

On choisit $w = 1$, $\phi_1 = 1$ et $\phi_2 = 1$.
On se place dans l'intervalle [-1000 ;1000].
On prend une population de 2000 et on étudie 1000 points, avec un nombre d'itérations maximales de 1000.



Test de validation





Etude paramétrique selon la taille de l'essaim

Ici, on fait varier nos expériences en changeant la taille de l'essaim. Pour cela, dans chaque expérience nous gardons w=1, ϕ_1=1 et ϕ_2=1 . Cependant, la taille de l'essaim varie de 100 à 1000.

Etude paramétrique selon la taille de l'essaim

Ici, on fait varier nos expériences en changeant la taille de l'essaim. Pour cela, dans chaque expérience nous gardons w=1, ϕ_1=1 et ϕ_2=1 . Cependant, la taille de l'essaim varie de 100 à 1000.

Taille de l'essaim	Xmin	f(Xmin)	Nombre d'itérations
100	[1.02070949 ; 1.08213535]	0.16273701	102
300	[1.27860654 ; 1.63176694]	0.07856271	70
500	[1.08377069 ; 1.17534695]	0.00707963	40
1000	[1.08307939 ; 1.16752211]	0.00997008	12

Observations de l'étude

Pour chaque test de notre expérience, nous avons gardé w=1, ϕ_1=1 et ϕ_2=1. Nous avons fait varier la taille de l'essaim de 100 à 1000. Nous avons pu observé que lorsque la taille de l'essaim augmentait, le nombre d'itérations diminuait.
Nous avons également observé que plus on augmentait la taille de l'essaim, plus la valeur de la fonction évaluée en notre Xmin (notre minimum) était proche de 0, qui est la valeur exacte minimale.
Nous sommes donc amenés à supposer que lorsque la taille de l'essaim augmente, les particules convergent plus rapidement vers l'optimum. On a également une meilleure précision pour celui-ci.

Etude paramétrique selon ϕ_2

Ici, on fait varier nos expériences en changeant notre valeur de ϕ_2. Pour cela, dans chaque expérience nous gardons w=1 et ϕ_1=1 et une taille de l'essaim égale à 500. Cependant, ϕ_2 varie de 0.5 à 1.5.

Etude paramétrique selon ϕ_2

Ici, on fait varier nos expériences en changeant notre valeur de ϕ_2. Pour cela, dans chaque expérience nous gardons w=1 et ϕ_1=1 et une taille de l'essaim égale à 500. Cependant, ϕ_2 varie de 0.5 à 1.5.

Valeur de ϕ_2	Xmin	f(Xmin)	Nombre d'itérations
0.5	[0.97033649 ; 0.94606621]	0.00291691	55
0.7	[0.94320374 ; 0.89711675]	0.00882603	59
0.8	[1.32667474 ; 1.71359502]	0.32267037	12
1.	[0.91747737 ; 0.81736255]	0.0663565	28
1.2	[1. ; 1.]	0.	4
1.5	[0.86023447 ; 0.74317104]	0.02053783	69

Observations de l'étude

Pour chaque test de notre expérience, nous avons gardé w=1 et ϕ_1=1 ainsi qu'une taille de l'essaim égale à 1000. Nous avons fait varier ϕ_2 de 0.5 à 1.5.
Nous avons pu observé qu'en augmentant ϕ_2 jusqu'à la valeur 1.2, la nombre d'itérations diminuait (observation globale, malgré quelques écarts).
Nous avons également observé qu'une fois que ϕ_2 avait dépassé la valeur 1.2, le nombre d'itérations augmentait à nouveau.
Nous sommes donc amenés à supposer que lorsque l'influence sociale augmente, les particules convergent plus rapidement vers l'optimum. Cependant, il existe une valeur limite pour cela. A partir de celle-ci, l'influence sociale devient trop importante par rapport aux autres influences pour une vitesse de convergence optimale.

Intérêts

Ce projet nous a permis de comprendre le fonctionnement de la PSO, méthode majeure utilisée dans de nombreux domaines, et susceptible de nous être utile au cours de notre vie professionnelle.

Intérêts

Ce projet nous a permis de comprendre le fonctionnement de la PSO, méthode majeure utilisée dans de nombreux domaines, et susceptible de nous être utile au cours de notre vie professionnelle.

Nous avons été particulièrement intéressés d'apprendre que des phénomènes naturels soient transformés sous forme mathématique afin d'être utilisés dans divers domaines.

Intérêts

Ce projet nous a permis de comprendre le fonctionnement de la PSO, méthode majeure utilisée dans de nombreux domaines, et susceptible de nous être utile au cours de notre vie professionnelle.

Nous avons été particulièrement intéressés d'apprendre que des phénomènes naturels soient transformés sous forme mathématique afin d'être utilisés dans divers domaines.

Il a également été intéressant de créer un algorithme permettant d'évaluer tous types de fonctions. Le fait de travailler sur un outil permettant de résoudre un large panel de problèmes suscite un intérêt particulier.

Problèmes rencontrés

Tout d'abord, nous possédions des méthodes fonctionnant bien individuellement mais qui, une fois combinées avec d'autres, causaient des erreurs.

Problèmes rencontrés

Tout d'abord, nous possédions des méthodes fonctionnant bien individuellement mais qui, une fois combinées avec d'autres, causaient des erreurs.

Nous nous sommes rendus compte que nous ne mémorisions pas assez d'éléments et de la mauvaise manière. En effet, lorsque nous mettions à jour nos nouveaux X, les anciens X que nous stockions se metaient également à jour. Nous avons donc dû utiliser deepcopy pour cela. Grâce à cela, nous faisions une véritable copie de l'élément et non pas un autre accès au même élément.

Problèmes rencontrés

Nous avons eu un problème lors de la création des constantes aléatoire U_1 et U_2. En effet, nous avons utilisé la fonction random de numpy : numpy.random.randn(1). Puis, lors des tests, nos valeurs étaient parfois abérrantes et nous avons vu que nos erreurs provenaient du random. Nous nous sommes donc tournés vers la fonction random.random() du module random.

Problèmes rencontrés

Nous avons eu un problème lors de la création des constantes aléatoire U_1 et U_2. En effet, nous avons utilisé la fonction random de numpy : numpy.random.randn(1). Puis, lors des tests, nos valeurs étaient parfois abérrantes et nous avons vu que nos erreurs provenaient du random. Nous nous sommes donc tournés vers la fonction random.random() du module random.

Nous avons finalement trouvé ce projet très intéressant avec, malgré des difficultés liées à la distance, une bonne cohésion d'équipe.

Annexe

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: Julien Choukroun & Samy David & Jessica Gourdon & Luc Sagnes
"""


import numpy as np
import random
from numpy.linalg import *
import scipy as sp
import matplotlib as mpl
import matplotlib.pyplot as plt
import math as mths
import copy as cp


borneInf = -1000
borneSup = 1000
Population = 2000
cas = 1000
dimension = 2
nbIteration = 1000

# Construit le nuage de points aléatoirement
def initialise(n, cas, borneInf, borneSup):
    x = np.zeros((n,cas))
    for i in range(n):
        for j in range(cas):
            x[i,j] = np.random.randint(borneInf, borneSup)
    return x

# Construction de la fonction test
def Rosenbrock(E):
    f = []
    n = np.shape(E)[0]
    for i in range(n-1):
        fi = 100*(E[1]-E[0]**2)**2+(1-E[0])**2
        f.append(fi)
    f=np.asarray(f)
    return f

# Calcule la différence des barycentres entre 2 points
def barycentre(xAncien, xNouveau, eps):
    baryAncien = np.zeros((1,np.shape(xAncien)[0]))
    baryNouveau = np.zeros((1,np.shape(xAncien)[0]))
    # On parcourt toutes les colonnes
    for i in range(np.shape(xAncien)[1]):
        baryAncien = baryAncien+xAncien[:,i]
```

Annexe

```
    for i in range(np.shape(xAncien)[1]):
        baryAncien = baryAncien+xAncien[:,i]
        baryNouveau = baryNouveau+xNouveau[:,i]
    baryAncien = 1.0*baryAncien/np.shape(xAncien)[1]
    baryNouveau = 1.0*baryNouveau/np.shape(xNouveau)[1]
    norme = mths.sqrt(np.sum((baryNouveau-baryAncien)**2))
    return norme>eps

def PSO(E, f, nbIteration, w, phi1, phi2):
    # Initialisation
    #nb : Nombre de x0 pris
    #V0 : vitesse initale de chaque particule (égale à 0) au début.
    #xAncien : nos xi tout au long de notre fonction
    #xLB : les local best de chaque particule
    #fLB : la fonction de Rosenbrock évaluée aux minimums locaux
    #fGB : la fonction de Rosenbrock évaluée à notre global best
    #xGB : notre minimum global
    nb = np.shape(E)[1]
    V0 = np.zeros(np.shape(E))
    xAncien = cp.deepcopy(E)
    xLB = cp.deepcopy(E)
    # On calcule l'image de xLB dans notre fonction
    fLB = f(xLB)
    # On cherche le minimum global xGB qui correspond au meilleur xLB
    fGB = np.min(fLB)
    index = np.where(fLB == fGB)
    index = index[1]
    xGB = xLB[:,index]
    # On fait les deux premières itérations de la boucle séparé pour pouvoir
    # nous servir de la différence de barycentre ensuite
    VAncien = V0
    # On calcule nos vitesses V_(i+1) puis nos positions x_(i+1)
    VSuiv = w*VAncien+phi2*random.random()*(xGB-x0)
    xSuiv = xAncien+VSuiv
    # On regarde ensuite si nos x_(i+1) sont mieux que nos xLB
    # Si tel est le cas, on change nos xLB des x_i en question
    for i in range(nb):
        fxi = f(xSuiv[:,i])
        if ((fxi < fLB).any()):
            xLB[:,i] = xSuiv[:,i]
    # On calcule l'image de xLB dans notre fonction
    fLB = f(xLB)
    # Si on a trouvé un nouveau global best, on le modifie avec l'ancien
    fGBNouveau = np.min(fLB)
    if (fGBNouveau < fGB):
        fGB = fGBNouveau
        index = np.where(fLB == fGB)
        index = index[1]
        xGB = xLB[:,index]
    # On continue ensuite jusqu'à atteindre le maximum d'itération ou de trouver
```

Annexe

```
94          index = np.where(fLB == fGB)
95          index = index[1]
96          xGB = xLB[:,index]
97      # On continue ensuite jusqu'à atteindre le maximum d'itération ou de trouver
98      # une distance entre 2 barycentres assez petite entre nos x_(i+1) et nos x_i
99      k = 1
100     while ((k <= nbIteration) and barycentre(xAncien, xSuiv, 10E-6)) :
101         VAncien = cp.deepcopy(VSuiv)
102         xAncien = cp.deepcopy(xSuiv)
103         # Dans notre cas on doit faire varier phi2
104         VSuiv = w*VAncien+phi1*random.random()*(xLB-xAncien)+phi2*random.random()*(xGB-xAncien)
105         xSuiv = xAncien+VSuiv
106         for i in range(nb):
107             fxi = f(xSuiv[:,i])
108             # On regarde si nos x_(i+1) sont mieux que nos xLB
109             # Si tel est le cas, on change nos xLB des x_i en question
110             if ((fxi < fLB).any()):
111                 xLB[:,i] = xSuiv[:,i]
112         # On calcule l'image de xLB dans notre fonction
113         fLB = f(xLB)
114         # Si on a trouvé un nouveau global best, on le modifie avec l'ancien
115         fGBNouveau=np.min(fLB)
116         if (fGBNouveau < fGB):
117             fGB = fGBNouveau
118             index = np.where(fLB == fGB)
119             index = index[1]
120             xGB = xLB[:,index]
121         k=k+1
122     plt.scatter(xLB[0,:],xLB[1,:],1,c='red')
123     plt.ylim(-3000,3000)
124     #plt.title('Essaim final')
125     plt.title('Essaim initial (bleu) et essaim final (rouge)')
126     print("Nombres itérations : ",k)
127     # Meilleure solution trouvée xGB
128     xMini = xGB
129     fMini=f(xMini)
130     return xMini,fMini
131
132
133 x0 = initialise(dimension, cas, borneInf, borneSup)
134 plt.scatter(x0[0,:],x0[1,:],1,c='blue')
135 #plt.title('Essaim initial')
136 v0 = np.zeros((dimension,cas))
137
138 f = Rosenbrock(x0)
139
140 xMini,fMini = PSO(x0, Rosenbrock, nbIteration, 1, 1, 1)
141 print("valeur de x min :",xMini)
142 print("Valeur de f min : ",fMini)
143
```