Uniwersytet Technologiczno-Przyrodniczy im. Jana i Jędrzeja Śniadeckich w Bydgoszczy
Wydział Telekomunikacji, Informatyki i Elektrotechniki

Antoni Malak
108650

Analiza oraz implementacja algorytmu PSO

Informatyka stosowana
sem IV
grupa 1

github.com/anteczko/PSO

Bydgoszcz 10.09.19

Historia

Algorytm PSO (particle swarm optymalization), został stworzony przez Dr.Kennedy'ego oraz Dr.Eberhart'a w roku 1995. Czerpał on inspirację z zachowania grup zwierząt, takich jak ptaki, ryby czy mrówki, szukających pożywienia. Mimo prostoty pojedynczych jednostek(agentów), oraz prymitywnej komunikacji, były w stanie znajdować cele, oraz informować siebie nawzajem o jego znalezieniu. Na początku algorytm miał symulować zachowania zwierząt, ale po jego uproszczeniu okazało się, że bardzo dobrze nadaje się do optymalizacji.

Znalazł zastosowanie w szukaniu m.i. ekstremów funkcji trójwymiarowych, oraz rozwiązywaniu bardziej złożonych, nieporstych problemów, które moża opisać funkcjami matematycznymi. Jego zaletą jest szybkość w znajdowaniu celów oraz prostota samego algorytmu, co się przekłada na jego prostą implementację. Czasami jego losowość i nieprzewidywalność może być przydatną cechą.

Opis i działanie

Algorytm pracuje za pomocą roju n cząsteczek(agentów) gdzie każda z nich może być potencjalnym rozwiązaniem. Każda z nich oznaczona jest indeksem i, posiada pozycję x,y o podanym zakresie, oraz prędkość w danej osi x,y. Każda z nich sprawdza, oraz zapamiętuje wartość funkcji w swojej aktualnej pozycji. Dodatkowo każda z nich zapamiętuje najlepszą wartość funkcji oraz jej pozycję, jako maksimum lokalne.

Na początku pozycja oraz prędkość każdego punktu jest wylosowywana w taki sposób, aby wartości mieściły się w podanym zakresie, oraz dla pozycji każdego punktu jest obliczana wartość funkcji. Jeżeli jest ona 'personalnym rekordem' to pozycja i wartość jest zapisywana jako maksimum lokalne tego jednego punktu.

Do użycia algorytmu potrzebne jest również wyznaczenie maksimum globalnego z pośród wszystkich cząsteczek. Ustanawiane jest poprzez sprawdzenie wartości każdej cząsteczki z osobna, oraz przyrównania go do maksimum globalnego. Jego pozycja maksimum globalnego również jest zapamiętywana.

Następnie prędkość dla danej osi x.y jest obliczana według poniższego wzoru:

$$v_i = W v_0 + c_1 rand(x_l - x) + c_2 rand(x_g - x)$$

$v_i - nowa\ prędkość\ cząsteczki\ o\ indeksie\ i$
$v_0 - akualna\ prędkość\ cząsteczki\ o\ indeksie\ i$
$x - aktualna\ pozycja\ cząsteczki$
$x_l - maksimum\ lokalne$
$x_g - maksimum\ globalne$
$rand - liczba\ losowa \in \langle 0; 1 \rangle$
$W - waga\ określająca\ zachowanie\ prędkości$
$c_1 / c_2 - wagi\ do\ modyfikacji\ zachowania\ cząsteczek$

Gdzie x jest zastępowany przez y, oraz funkcja jest obliczana osobno dla każdej cząsteczki.

Za pomocą zmiany wartości C1 oraz C2 można zmienaić zachowanie stadne cząsteczek. Wieksza wartość C1 bardziej zbliża cząśteczki do maksimum globalnych, a większa wartość C2 bardziej je kieruje w kierunku maksimum globalnego. Zbyt duże te wartości mogą sprawić, że punkty skupią się w maksimum lokalnym, oraz nie znajdą maksimum globalnego ze względu na zbyt duże przyciąganie do lokalnego oraz brak rozbieżności.

Następnie pozycja każdej z cząsteczek jest akualizowana wg. wzoru:

$$p = p_0 + v$$

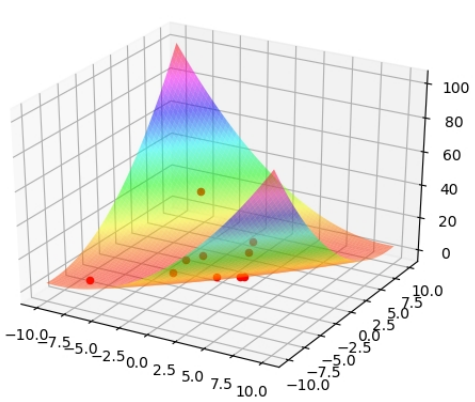
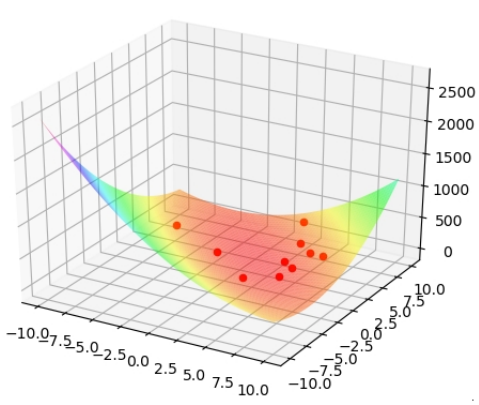
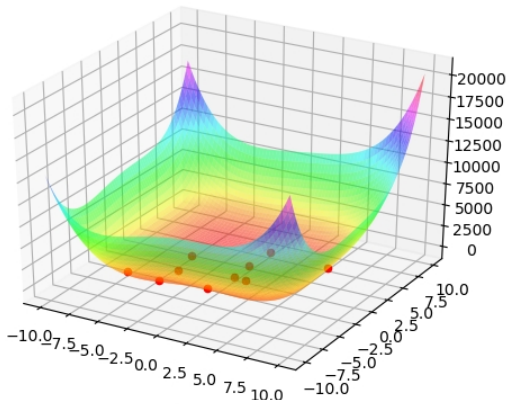
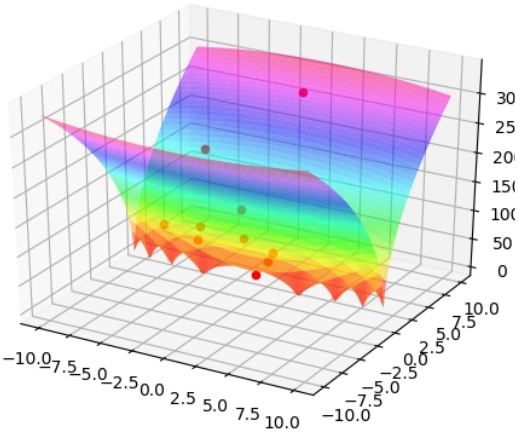
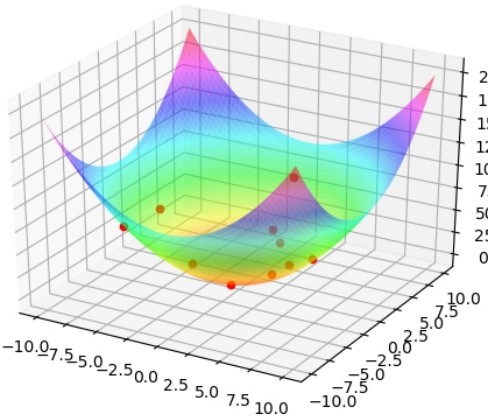
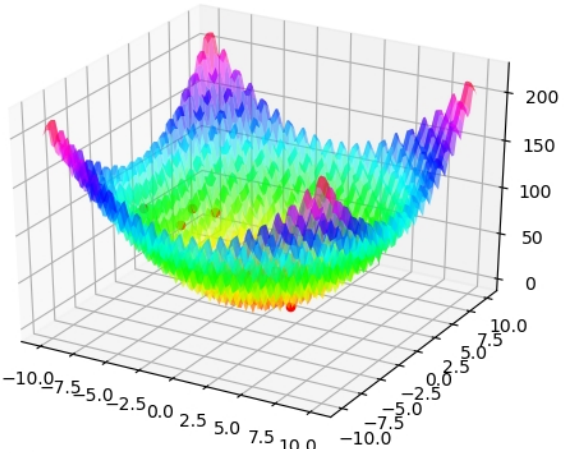
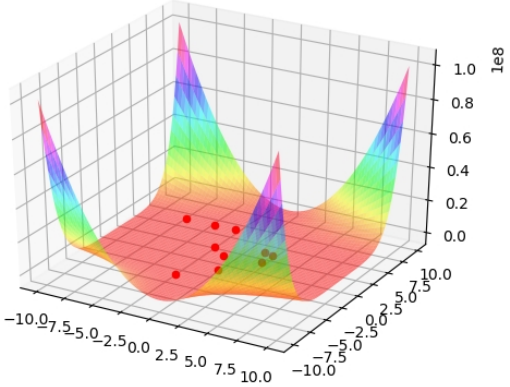
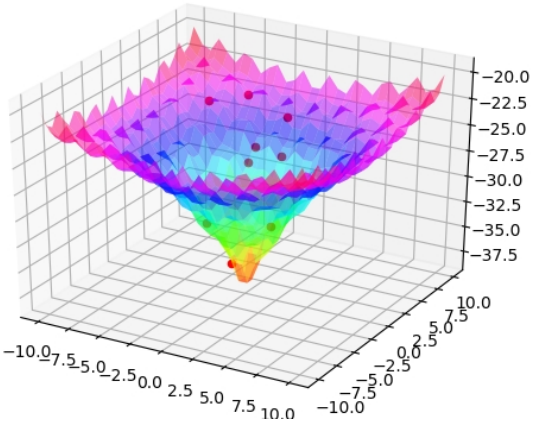
Czynność jest powtarzana dla każdej cząsteczki oraz cały proces jest powtarzany do wykonania maksymalnej liczby powtórzeń, albo dopóki nowo obliczone maksimum globalne nie różni się do poprzedniego o więcej niż z góry ustaloną małą wartość błędu.

Implementacja oraz zastowowane biblioteki

Do implementacji algorytmu w projekcie użyto języka python, ze względu na jego prostotę, szybkość oraz wygodę pisania. Dodatkowo do wizualizacji funkcji trójwymairowych oraz poszczególnych punktów użyto bilbioteki matplotlib.

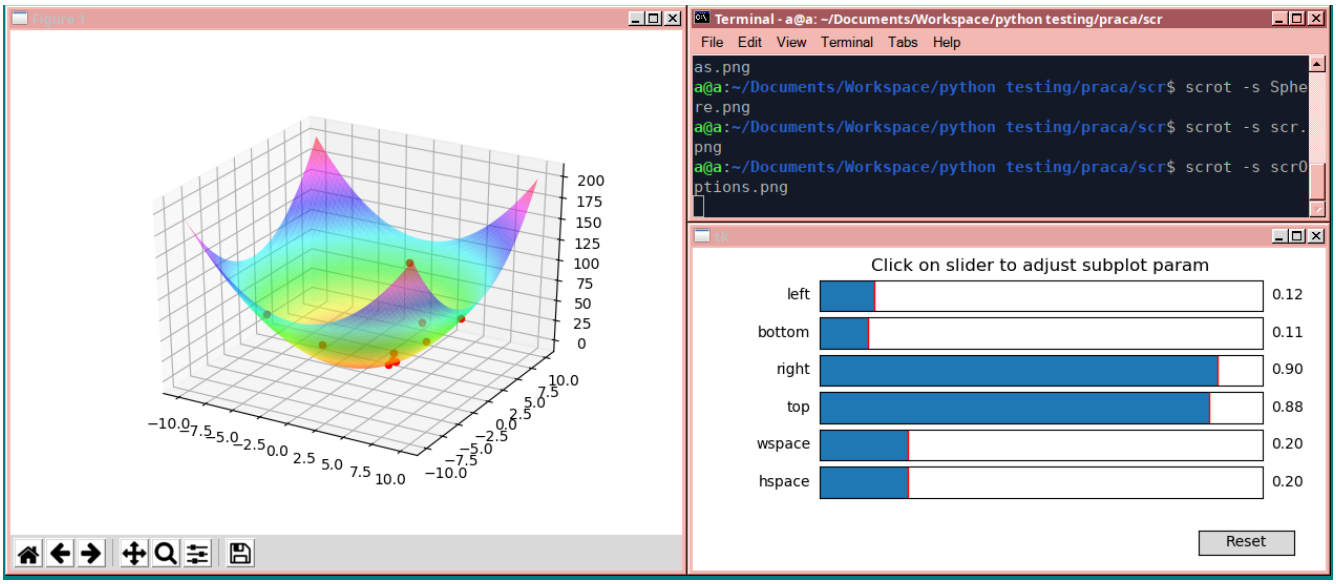
Program nie posiada interfejsu użytkownika, jedynym okienkiem jest okno bliblioteki wizualizujące fukcję oraz cząsteczki. Programu uruchamia się za pomocą komendy terminala. Za pomocą ustawienia zmiennej BENCHAMRKMODE można włączyć wizualizację rozwiązania funkcji, albo włączyć tryb mający zapewnić największą prędkość oraz jak najmniej opóźnień, przeznaczony do sprawdzania wydajności algorytmu. Niestety porównywanie ze sobą wyników różnych funckcji jest niemożliwe ze względu na różny stopień ich skomplikowania. Niektóre z nich zostały pomnożone przez minus, żeby algorytm zawsze szukał ich wartości minimalnej. Oto lista zaimplementowanych funkcji:

1. funkcja Matyasa $0.26\left(x^2+y^2\right)-0.48\,xy$
2. funkcja Bootha $\left(x+2\,y-7\right)^2+\left(x2+y-5\right)^2$
3. funkcja Himmeleblausa $\left(x^2+y-11\right)^2+\left(x+y^2-7\right)^2$
4. funkcja Bukina numer 6 $100\sqrt{\left|y-0.01\,x^2\right|}+0.01\left|x+10\right|$
5. funkcja Sferyczna x^2+y^2
6. funkcja Rastrigina $-\left(20+\left(x^2-10\cos\left(2\,\pi x\right)\right)\right)+\left(y^2-10\cos\left(2\,\pi y\right)\right)$
7. funkcja Bealea $\left(1.5-x+xy\right)^2+\left(2.25-x+xy^2\right)^2+\left(2.625-xy^3\right)^2$
8. funkcja Ackleya $-20\exp\left(-0.2\left[-0.2\sqrt{0.5\left(x^2+y^2\right)}\right]\right)-\exp\left[\left(0.5\left(\cos 2\,\pi x+\cos 2\,\pi y\right)\right)\right]+e+20$

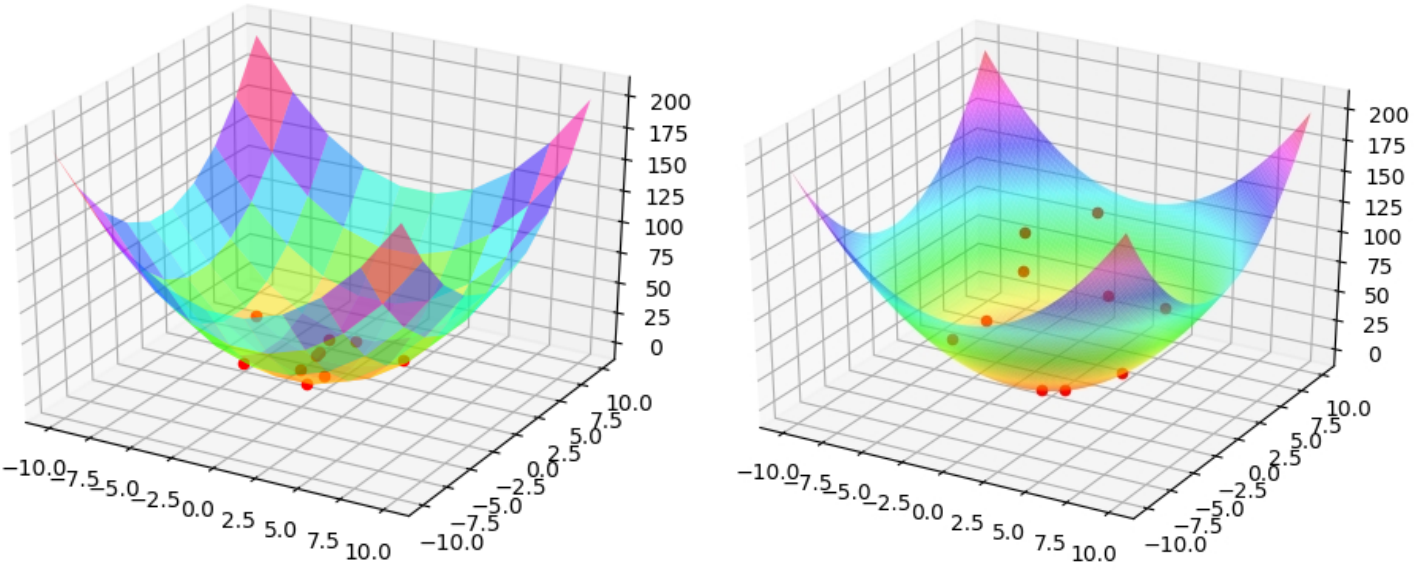









Zrzuty programu

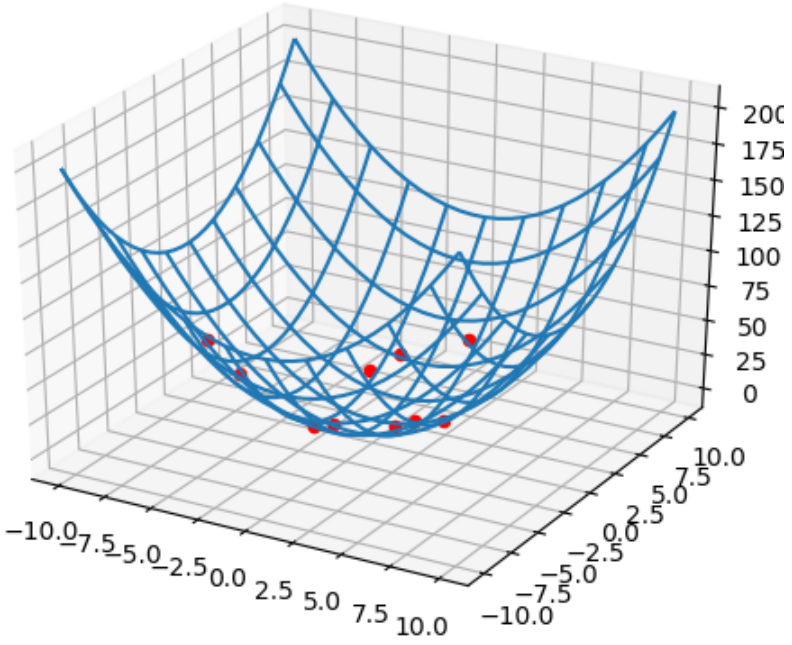
Program pod względem graficznym oferuje wszystkie opcje zapewnione przez bibliotekę matplotlib. Widok funkcji można obracać, przybliżać oraz można edytować wypełnienie obszaru okna przez wykres. Dodatkowo można zobaczyć jak punkty przesuwają się po funkcji gdy naciśniemy dowolny klawisz.



Można również zmieniać rozdzielczość narysowanej funkcji oraz zmieniać jej styl.



po lewej - rozdzielczość 10, po prawej 100



Kod programu

```
import random
import numpy as np
import time
import sys
import math
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from matplotlib.pyplot import plot, ion, show
from mpl_toolkits.mplot3d import axes3d
import argparse


random.seed(time.clock())
#CONSTANTS VALUES
W=0.5
C1=0.8
C2=0.9

RANGE=10
VELMAX=1
PARTICLESN=50

ERROR=1e-6

GRAPHRES=70
##Endof CONSTANT VALUES

#global values
BENCHMARKMODE=0
MEASUREMOD=100 #how many tests we are going to make before measuring time
LOOPS=1

ANIMATIONTIME=0.1
#end of global values

def printf(format, *args):
    sys.stdout.write(format % args)

def f(x,y):
    #return x**2+y**2 #sphere function
    #return 0.26*(x*x+y*y)-0.48*x*y #Matyas function
    #return pow((x+2*y-7),2)+pow((x*2+y-5),2) #Booth function
    #return pow((pow(x,2)+y-11),2)+pow((x+pow(y,2)-7),2) #Himmelblau's function
    #return ( (1+pow(x+y+1,2)) * (19-14*x+3**x-14*y+6*x*y+3**y) )*( (30+pow(2*x-3*y,2))*(18-32*x+12**x+48*y-36*x*y+27**y) )
    #return (100*np.sqrt(np.abs(y-(0.01*pow(x,2)))))+(0.01*np.abs(x+10))#Bukin function N.6
    #return -x**2-y**2-((np.abs(x)+np.abs(y))*(np.abs(x)+np.abs(y))) #sphere function
    #return 0.5+( (pow(x*x-y*y,2)-0.5)/pow((1+0.001*(x*x+y*y)),2) )
    #return ((10 * 2 + (x * x - 10 * np.cos(2 * math.pi * x)) + (y * y - 10 * np.cos(2 * math.pi * y)))) #Rastrigin fitness_function
    #return  ((pow(1.5 - x + x * y, 2) + pow(2.25 - x + x * y * y, 2) +pow(2.625 - x + x * y * y * y, 2)) )#Beale's function
    #return -20 * np.exp(-0.2 * np.sqrt(0.5 * (x * x + y * y))) + np.exp(0.5 * (np.cos(2 * math.pi * x) + np.cos(2 * math.pi * y))) - 20 #Ackley's fitness_function


def rand():
    return random.random()*2*RANGE-RANGE

class Best:
    x=0
    y=0
    fitness=0
    delta=0

    def __init__(self):
        self.x=rand()
        self.y=rand()
        self.fitness=f(self.x,self.y)
        self.delta=1000

    def printInfo(self):
        printf("x:%.2f y:%.2f gbFitness:%.2f delta:%f BEST!\n",self.x,self.y,self.fitness,self.delta)
```

```
Best=Best()

class Particle:
    x=0
    y=0
    xvel=0
    yvel=0
    fitness=f(x,y)
    bestFitness=f(x,y)
    bestX=x
    bestY=y

    def __init__(self):
        self.x=random.random()*2*RANGE-RANGE
        self.y=random.random()*2*RANGE-RANGE
        self.xvel=(random.random()*VELMAX)%(RANGE-self.x)
        self.yvel=(random.random()*VELMAX)%(RANGE-self.y)
        self.fitness=f(self.x,self.y)
        self.bestFitness=self.fitness
        self.bestX=self.x
        self.bestY=self.y

    def printInfo(self):
        printf("x:%.2f y:%.2f xv:%.2f yv:%.2f fit:%.2f pbfit:%.2f\n",self.x,self.y,self.xvel,self.yvel,self.fitness,self.bestFitness)

    def updateFitness(self):
        self.fitness=f(self.x,self.y)
        if(self.fitness<self.bestFitness):
            self.bestFitness=self.fitness
            self.bestX=self.x
            self.bestY=self.y
        if(self.fitness<Best.fitness):
            Best.delta=abs(Best.fitness-self.fitness)
            Best.fitness=self.fitness
            Best.x=self.x
            Best.y=self.y

    def updatePosition(self):
        self.xvel=W*self.xvel+(C1*random.random()*(self.bestX-self.x))+(C2*random.random()*(Best.x-self.x))
        self.yvel=W*self.yvel+(C1*random.random()*(self.bestX-self.x))+(C2*random.random()*(Best.y-self.y))
        self.x+=self.xvel
        self.y+=self.yvel


#initializing for every simulation
p=[]
#init particles
for i in range(PARTICLESN):
    p.append(Particle())
#ENDOF initializing for every simulation

if not(BENCHMARKMODE):
    #initializing for every simulation
    p=[]
    #init particles
    for i in range(PARTICLESN):
        p.append(Particle())
    #ENDOF initializing for every simulation

    print("Drawing everything!")
    #will print function and every point

    #create canvas
    fig = plt.figure()
    ax = plt.axes(projection='3d')

    # Grab some test data.
    x = np.linspace(-RANGE, RANGE, GRAPHRES)
    y = np.linspace(-RANGE, RANGE, GRAPHRES)

    X, Y = np.meshgrid(x, y)
    Z = f(X, Y)
    #interactive mode on
    plt.ion()

    j=0
    split=time.time()
```

```
    while (Best.delta>ERROR and j<100):
        ax.plot_surface(X, Y, Z,cmap=cm.hsv,lw=3,linewidth=1,alpha=0.5,rstride=1, cstride=1, )
        for i in range(PARTICLESN):
            p[i].updateFitness()
            p[i].updatePosition()
    #         p[i].printInfo()
            p[i].printInfo()
            ax.scatter3D(p[i].x, p[i].y,f(p[i].x,p[i].y),lw='1',color='r' )
        plt.show()
        raw_input()
        ax.clear()
        j+=1;

    print("at iteration ",j," time:",time.time()-split)
    Best.printInfo()

else:
    print("Gotta go fast!")
    split=time.time()
    for a in range(10000):
        #initializing for every simulation
        p=[]
        #init particles
        for i in range(PARTICLESN):
            p.append(Particle())
        #ENDOF initializing for every simulation

        j=0

        while (Best.delta>ERROR and j<10000):
            for i in range(PARTICLESN):
                p[i].updateFitness()
                p[i].updatePosition()
            j+=1
        #print("at iteration ",i," time:",time.time()-split)
        Best.printInfo()
    print(time.time()-split)
```

Źródła

http://www.alife.pl/optymalizacja-rojem-czastek
https://link.springer.com/article/10.1007/s00500-016-2383-8#Sec2
https://www.ii.uni.wroc.pl/~prz/2011lato/ah/opracowania/roj_czast.opr.pdf
http://zeszyty-naukowe.wwsi.edu.pl/zeszyty/zeszyt13/Zastosowanie_algorytmu_optymalizacji_rojem_czastek_do_znajdowania_ekstremow_globalnych_wybranych_funkcji_%20testowych.pdf
http://aragorn.pb.bialystok.pl/~wkwedlo/EA6.pdf