End of semester project

Global Optimization algorithms

Ecole Polytechnique de l'Université de Nice Sohia Antipolis
Applied Mathematics and Modelisation Department

June 2012

Teacher:
Pierre DREYFUSS

Students:

AGOSTINI Théo
MBYAS SAROUKOU Hassane
ZHANG Xuan

Table des matières

I. Introduction

Optimization aims,generally, to find the best solution called optimum of a problem by using a set of numeric methods. In this case, we are interested in algorithms solving optimization problems for real, continuous, differentiable and non-linear functions.

Several approachs are available, there are local methods giving a local optimum and global ones permiting to find a global optimum. Here, we will seek for minima because a maximzation problem can be considered as a minimization question.

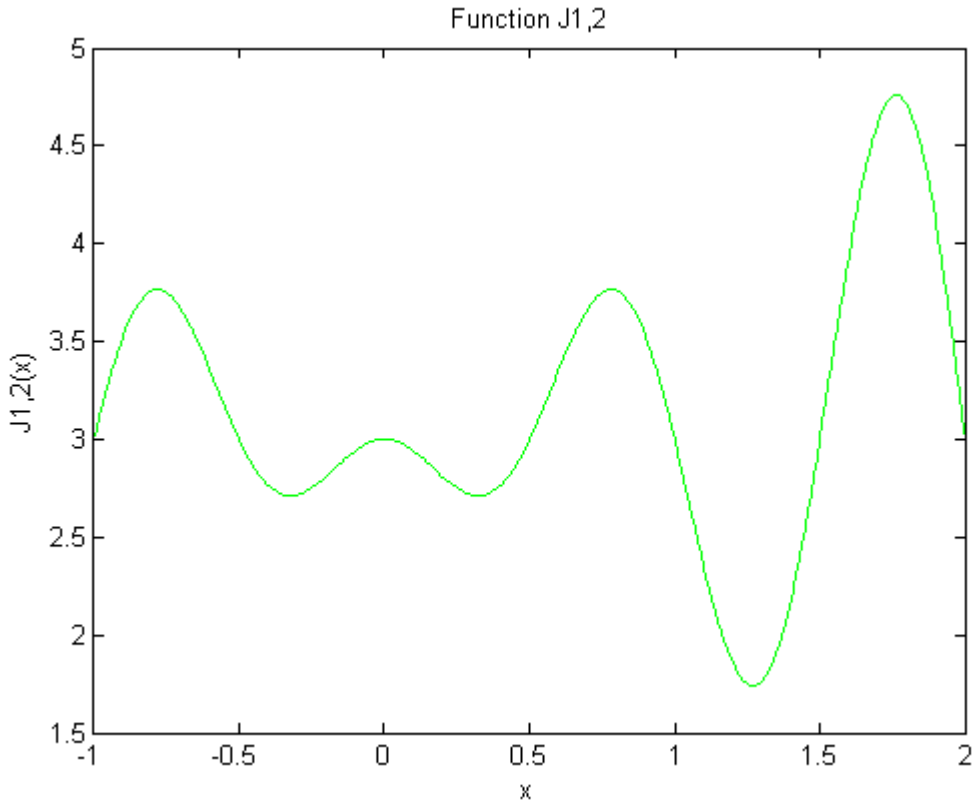
We will study global optimization algorithms, in particular: simulated annealing, ant colony and particle swarm.

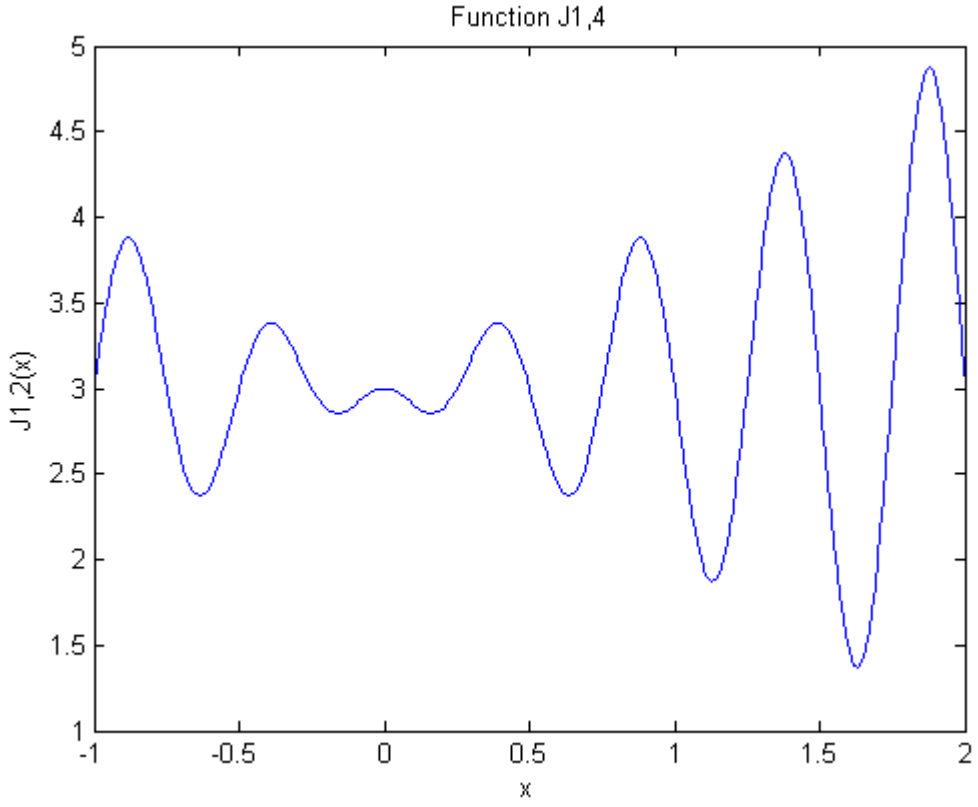
In the following, we search a global minimum of these functions:

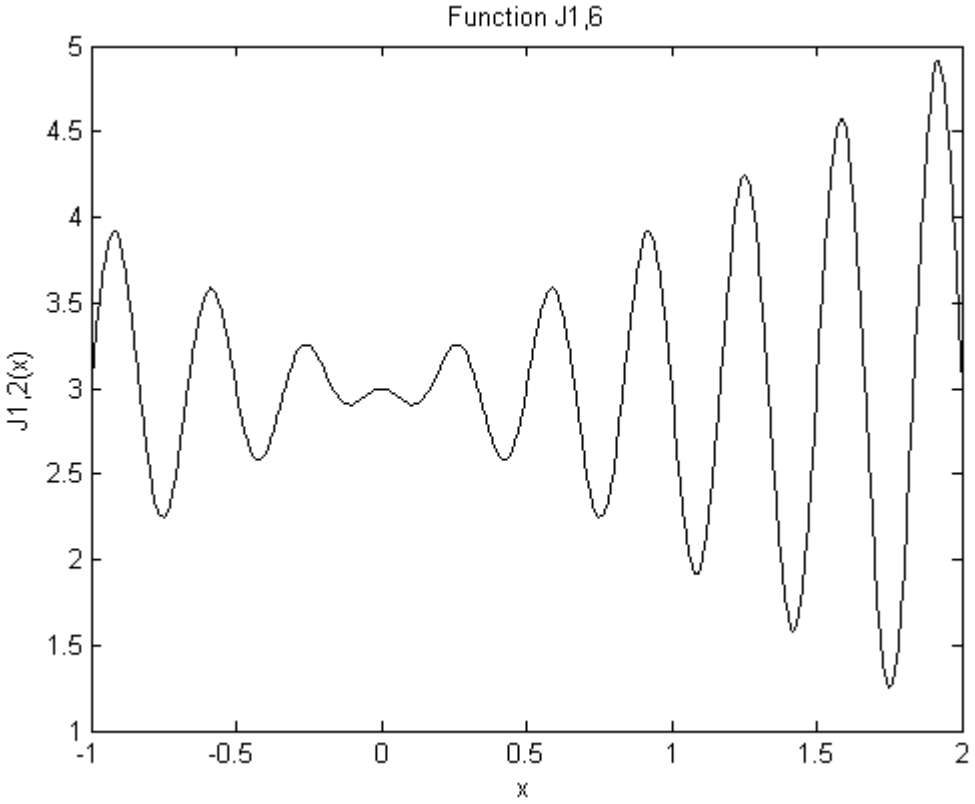
$J_{1,\omega}(x)=3-x\sin(\omega\pi x)$, $x\in[-1;2]$ and $\omega=2,4,\ldots,10$.

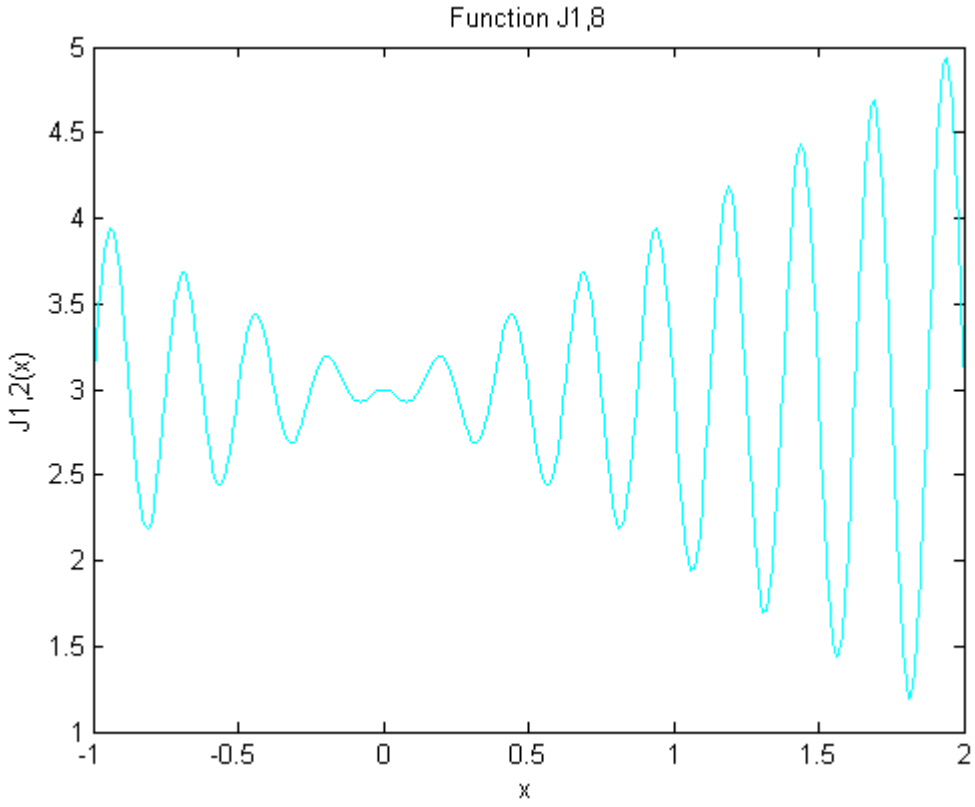
$J_2(x)=10\sin(0{,}3\,x\sin(1{,}3x^2+0{,}00001\,x^4+0{,}2\,x+80))$ with $x\in[-10;10]$.

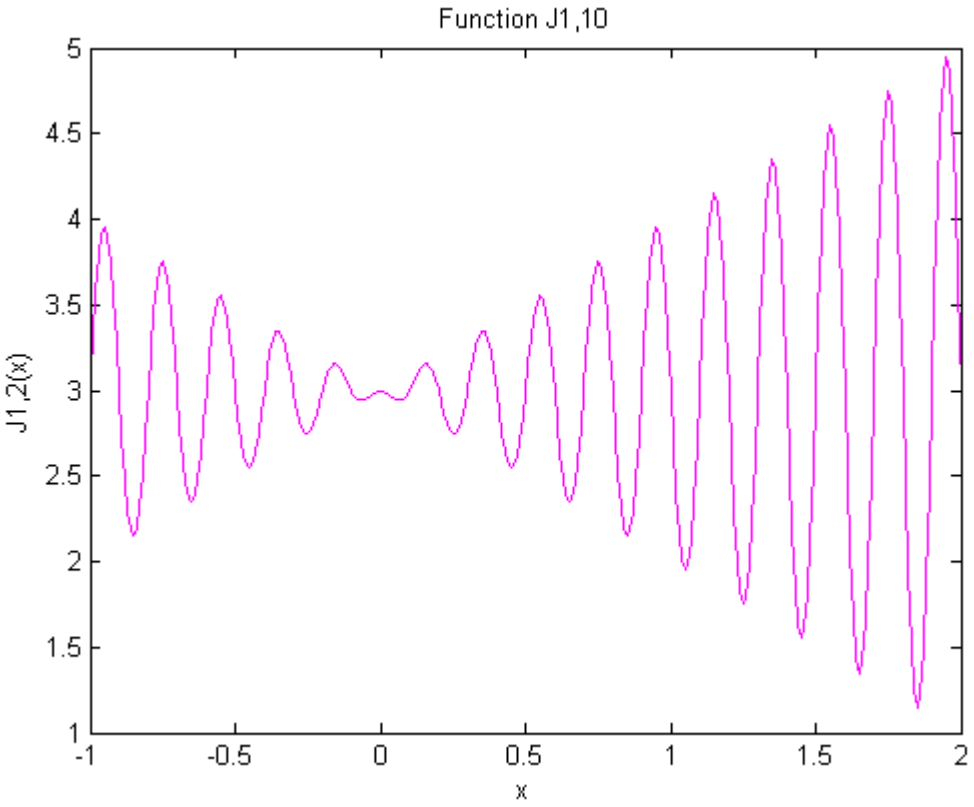
Graphs:

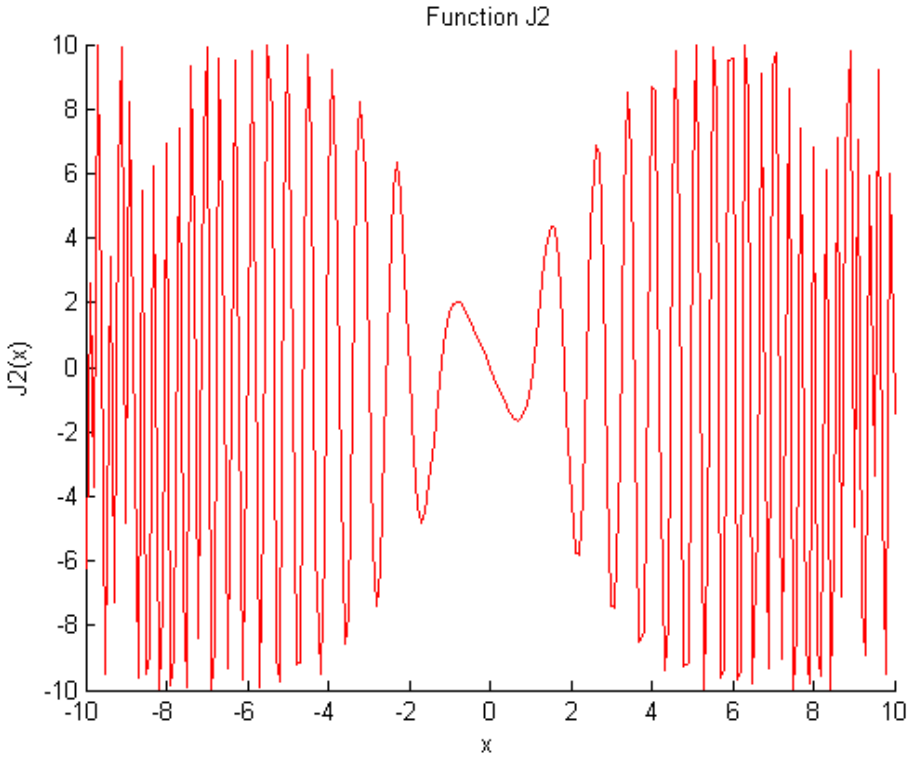





Function J1,6
J1,2(x)
x
5
4.5
4
3.5
3
2.5
2
1.5
1
-1
-0.5
0
0.5
1
1.5
2


Function J1,8
J1,2(x)
x
5
4.5
4
3.5
3
2.5
2
1.5
1
-1
-0.5
0
0.5
1
1.5
2


Function J1,10
J1,2(x)
x
5
4.5
4
3.5
3
2.5
2
1.5
1
-1
-0.5
0
0.5
1
1.5
2


Function J2
J2(x)
x
10
8
6
4
2
0
-2
-4
-6
-8
-10
-10
-8
-6
-4
-2
0
2
4
6
8
10

II. Simulated annealing algorithm (SA)

1.History

The method of annealing comes from the observation that the natural cooling of certain metals does not allow atoms to move into the configuration strongest.

The most stable configuration is reached by controlling the cooling and slowing it down with a supply of external heat.

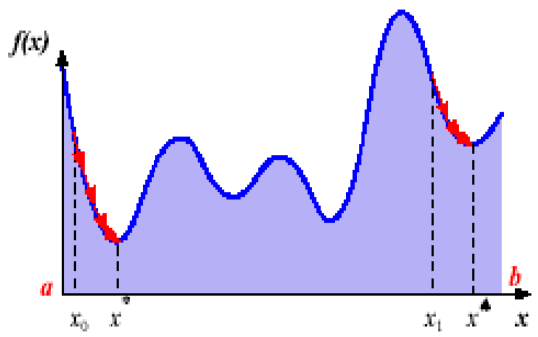
This process is used in metallurgy to improve the quality of a solid, we bring energy to fuse the metal. Then, by cooling the metal with the way more or less rapid, we seek for reaching a minimal energy state (which corresponds to a solid structure). We seek the most stable structure possible for this metal.

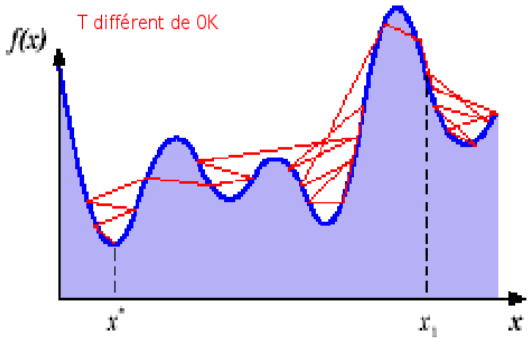
The annealing method was simulated (by Metropolis) in the 50s, the main idea is to move for finding the global minimum by choosing a neighbour point randomly. Then, if the value of the function is smaller at the neighbour point (at the current temperature), we will move there. Otherwise, we will move there with a certain probability (which decreases the value of the function which sign temperature). Finally, we will repeate this process by decreasing the temperature.

Furthermore, this algorithm was used for solving combinatorial optimization problems in the 80s (by Krickpatrick and Cerny): Suppose a company wants to distribute N different tasks (for example related to the manufacture, receipt of components, dispatching of products, accounting, etc..) in N local building. Each division has its advantages and disadvantages by regarding the match between the task and the local. We further assume that we can quantify, or at least classify the effectiveness of each way of the distributions in terms of cost of manufactured goods for example. The problem is to determine the most effective allocation without trying them all!

2.Principle,algorithm and choice of parameters

Principe :





First, take a starting point, initialize the energy and the temperature.
Next, at each iteration, we choose randomly a point close to the first one and compute the energy at this point. If the energy at this point is lower than before, we apply it to the current solution. Otherwise, it can be accepted or rejected according to the Metropolis rule.
Finally, temperature is decreased and process is repeated again until maximum number of iterations is reached.

Quasi-code of the algorithm:

```
s := s0
e := E(s)
T=Tmax
nIter := 0
while nIter < nIterMax and e > eMax
  sn := neighbour(s)
  en := E(sn)
  if en < e or MetropolisRule then
  s := sn; e := en
  nIter := nIter + 1
  T=reduction(T)
return s
```

Metropolis Rule :Define the probability that accept a non-optimal solution (hoping to manage to a lower minimum) : $e^{\frac{-\Delta E}{T}}$.
The fact that unminimizing solution may be chosen guarantee not to fall in local minimum.

Temperature Reduction :

Two main ways :

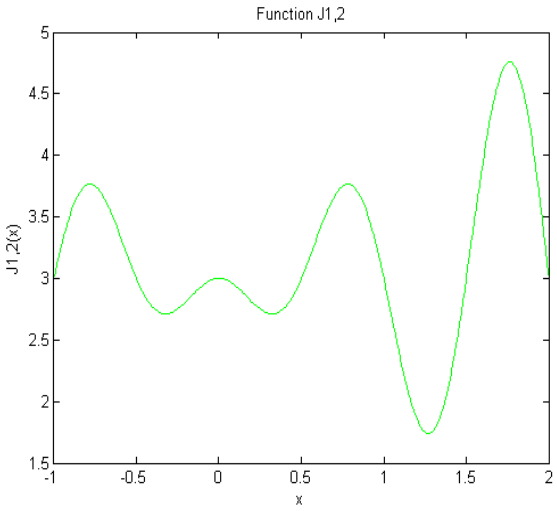
- Iterate with constant temperature until energy seems to be in a local minimum. Then, decrease temperature.
- Decrease continuously temperature with a continuous rule, for example newTemperature=0.99currentTemperature.

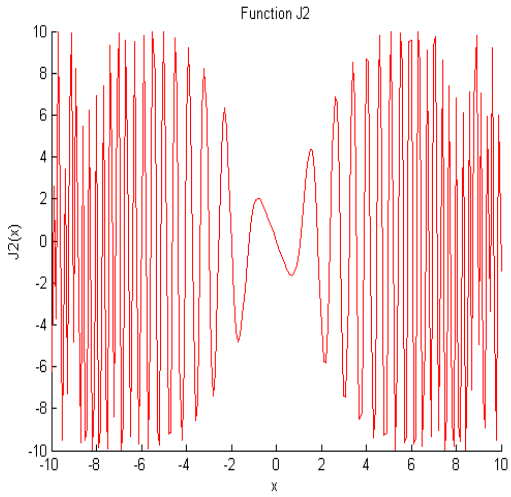
The way to reduce temperature is empirical choice.Indeed, we could find a link between the movement number, interval size and movement size. However, tests show that a slow continuous reduction is effective if the function doesn't fluctuate too much.

3.Applications and results

1D-functions :

In a space of dimension 1





Function J1,2: $J_{1,\omega}(x)=3-x\sin(\omega\pi x)$

The algorithm converges in 2500 iterations ,the computing time is 0.01seconds and the minimum global found is x=1,28.

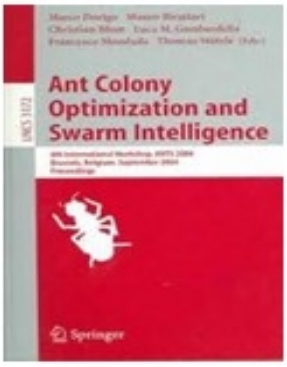
Function J2: $J_2(x)=10\sin(0{,}3\,x\sin(1{,}3x^2+0{,}00001\,x^4+0{,}2\,x+80))$

The algorithm converges in 2500 iterations ,the computing time is 0,01seconds and the minimum global found is is found is x= 9.8.

III. Ant colony optimisation algorithm

1.History

Proposed by Marco Dorigo in the 90S, the algorithm was inspired by the comportement of a group of ants for finding the best way between the ants and the food.
This method aims to construct the best solutions. Every time an individual find a solution of the problme, whatever good or bad, it will enrich the collective knowledge of the group. Furthermore, every time a new individual chosse the way, it will rely on the collective knowledge to make the choice.

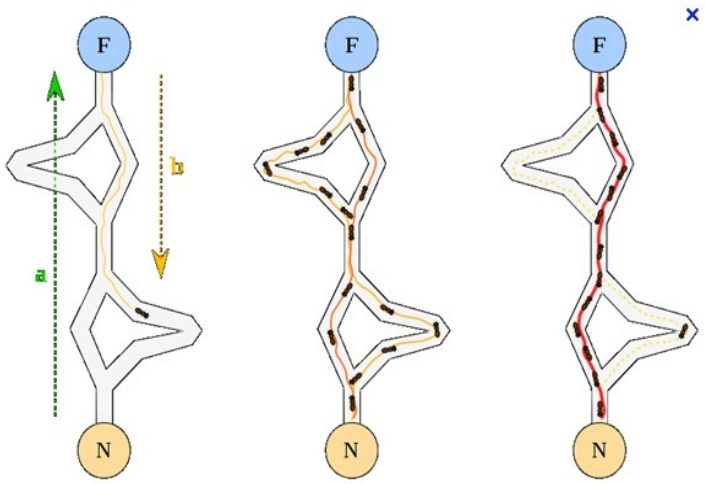


Biographer:
Marco Dorigo (born 26 August 1961 in Milan, Italy) is a research director for the Belgian Funds for Scientific Research (FNRS) and a co-director of IRDIA, the artificial intelligence lab of the Université libre de Bruxelles. He is the proponent of the "ant colony optimization" metaheuristic and one of the founders of the swarm intelligence research.

2.Ant colony optimization algorithm (ACO) for resolving discrete problem

Here, we take the Le problème NP complet (voyageur de commerce) for an example, the other discrete problems have exactly the same principle. The problem is that We have n cities, we want to travel all of them following a shortest path.

2.1 Principle,algorithm and choice of parameters



The mechanism of this algorithm comes from the behavior of the ant colony

<u>Step 1</u> -Initialize the distribution of the ant colony
we put m ants in the n cities randomly.

<u>Step 2</u> -Determine the rule of move of the ant colony

The k[th] ant in the city i will move to the next city with the probability as below:

$$P^k(i,j) = \begin{cases} \dfrac{[\tau(i,j)]^{\alpha} \cdot [\eta(i,j)]^{\beta}}{\sum_{s \notin tabu_k} [\tau(i,s)]^{\alpha} \cdot [\eta(i,s)]^{\beta}}, & if\ j \notin tabu_k \\ 0, & otherwise \end{cases} \quad (1)$$

$\iota(i,j)$ signs the density of pheromone (the marker of ants, more ants pass a path, more pheromone they will leave).

$\eta(i,j)=1/d(i,j)$ is the information of inspiration with d(i, j) is the distance between city i and city j.

α and β reflect the the relative importance of the pheromone and the information of inspiration.

$tabu_k$ is the list of cities which the ants have already visited.

When all the ants have finished this travel round (this iteration), the pheromone will be updated as follows:

$$\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}$$

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \qquad (2)$$

ρ is the parameter which indicates the persistence of pheromone, $\rho < 1$.

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} & ij \in l_k \\ 0 & otherwise \end{cases} \qquad (3)$$

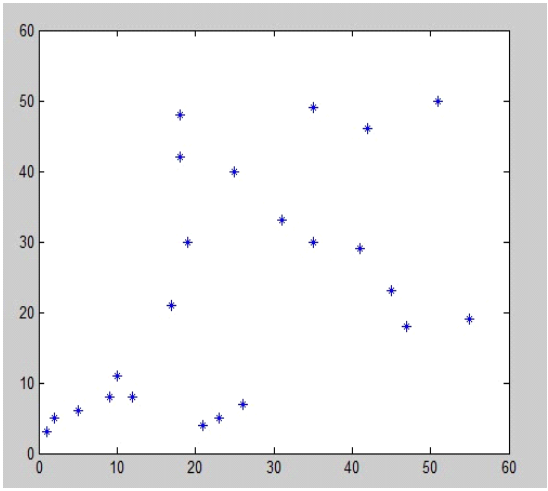
is the quantity of pheromone which the ant a_k leaves when it moves from city i to city j. Q is a constant and l_k is the path of ant a_k in this iteration, L_k is its length.

2.2 The condition to stop

(1) fix a number of times maximal of the iteration.
(2)the current solution dosn't change for successive k times, which means that the algorithm has already converged.

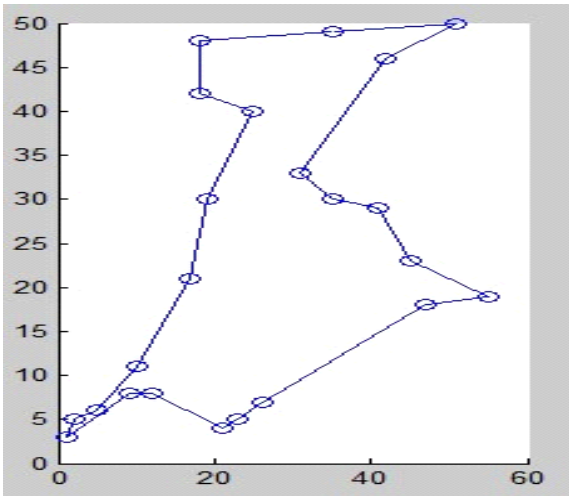
2.3 Applications and results

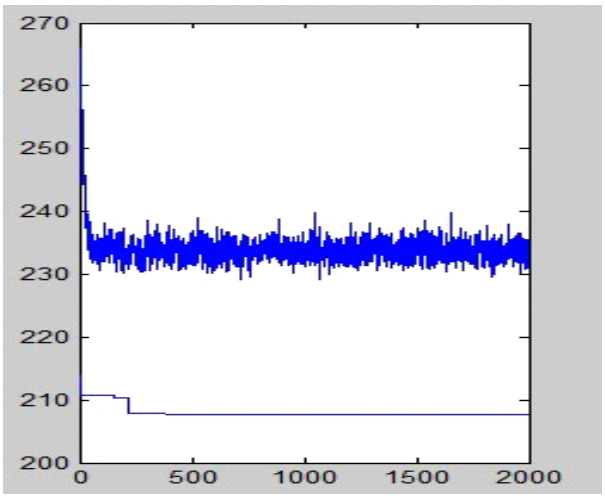
An example for test: we have 23 cities which distribute randomly as follows:



We put the coordinates of these 23 cities in the matrix C, in this case,
C=[1,3;2,5;9,8;10,11;21,4;5,6;23,5;31,33;25,40;12,8;26,7;18,48;51,50;55,19;35,30;42,46;45,23;41,29;19,30;18,42;17,21;35,49;47,18] and the indice of city is [1,2,...,23]
Which means the coordinate of city 1 is (1,3), and that of city 2 is (2,5), etc.

We solve the problem with the ant colony algorithm in matlab, after 230 iterations, the solution converge and we obtain the result as below:





The shortest path is
Shortest_Route =
[12 20 9 19 21 4 6 2 1 3 10 5 7 11 23 14 17 18 15 8 16 13 22]
and the shortest length is 207.7338.

3.Ant colony optimization algorithm (ACO) for resolving continuous function with multiple extremums

3.1 Principle,algorithm and choice of parameters

Principle: Ant colony optimization algorithm (ACO) for resolving continuous function with multiple extremums

The behavior of a single individual of ants is very simple, however after forming a group, they can show an amazing organizational behavior. Ants have a very important feature, that is in the process of searching for food, they can always find the shortest path leading between the source of food and their nests. The ant colony optimization algorithm is based on the behavior of ants searching for food in nature.

We take the one dimension function as an example and we search for its minimums. Suppose the function y=f(x) is defined in interval [a, b]. Firstly, we divide the interval into n small subinterval with the same size, marked as $\{I_1, I_2, \ldots, I_n\}$ an the midpoint of I_i is noted as x_i .Assuming that the neighbour intervals of I_i are I_{i-1} and I_{i+1} , and suppose that there is an virtual edge $e(I_i, I_{i+1})$ between I_i and I_{i+1} , the weight of this edge (the virtual distance) is related to the value of $f(x_i)-f(x_{i+1})$, the bigger the value of $f(x_i)-f(x_{i+1})$, the more possible that the ants will move from the interval I_i to the interval I_{i+1} . After the ants moving to the interval I_{i+1} , they will leave the pheromone, the quantity of the pheromone is also related to $f(x_i)-f(x_{i+1})$, the more pheromone in the interval I_{i+1} , the more possible it will appeal to the ants to come.

After the ants moving, some of the intervals have many ants, the others have not any ant. The intervals which contain the ants are exactly where the extremums are. On the other hand, the intervals which have no ant will almost impossible contain the extremums. We take out the intervals which contain the ants and we refine these intervals again, and we repeat the search process above-mentioned until the refined intervals are enough small. Finally, all the ants stop at the points of extremums or nearby, the midpoints of the intervals where the ants stays are the extremums.

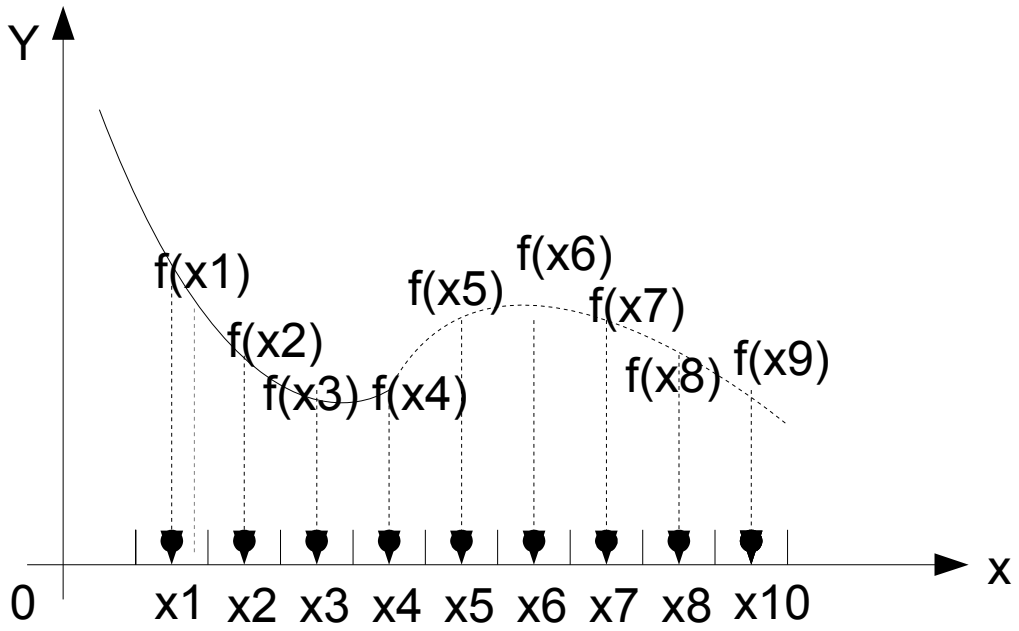
To sum up, the realization of the ant colony optimization algorithm consist of the 4 following steps:
Step 1 -Initialize the distribution of the ant colony
Step 2 -Determine the rule of move of the ant colony
Step 3 -Update of the pheromone
Step 4 -Narrow the research space

3.2 L'algorithm of realization

Step 1 -Initialize the distribution of the ant colony

we divide equally [a, b] into n subinterval, the length of each subinterval equals $\delta=\frac{b-a}{n}$, each subinterval is noted as $\{I_1, I_2, \ldots, I_n\}$, the i[th] subinterval $I_i=[a+(i-1)\delta, a+i\delta]$ and its midpoint $x_i=a+(i-\frac{1}{2})\delta$. At the very beginning moment, we put an ant at the midpoint of each subinterval and they are noterd respectively as $a_1, a_2, \ldots, a_n$, each ant a_i has a function value $f(x_i)$. We note also $\iota_i(t)$ as the pheromone of subinterval I_i at the moment t. At the first moment, each subinterval has the same quantity of pheromone, which means $\iota_i(0)=const$ (const is a positive constant), and we initialize also the increment of pheromone $\Delta\iota_i=0, \quad (i=1,\ldots,n)$.

Graph 1: the initial distribution of the ant colony.



<u>Step 2</u> -Determine the rule of move of the ant colony

We note $neighbor(I_i)$ as the set of neighborhood of I_i , for the one dimension function we have then : $neighbor(I_i)=\begin{cases} I_{i+1} & i=1 \\ \{I_{i-1}, I_{i+1}\} & i=2,\ldots,n-1 \\ I_{i-1} & i=n \end{cases}$. The ant at the subinterval I_i will move to its neighborhood according to the weight of virtual edge $e(I_i, I_{i+1})$: $f(x_i)-f(x_{i+1})$, we define the function of inspiration $\eta_{ij}=|f(x_i)-f(x_{i+1})|$. We suppose that the ant a_k was currently at the subinterval I_i , if $f(x_i)-f(x_{i+1})>0$, the ant a_k will move to its neighbor I_j ; otherwise, it will not move. We note $allowed_k$ as the set of subintervals where the ant a_k can move to next step and $p_{ij}^{(k)}(t)$ as the probability with which the ant a_k move from subinterval I_i to I_j at the t[th] iteration. We have the expression of $p_{ij}^{(k)}(t)$ as follows:

$$p_{ij}^{(k)}(t)=\begin{cases}\dfrac{\iota_j^{\alpha}(t)\eta_{ij}^{\beta}}{\sum_{h\in allowed_k}\iota_h^{\alpha}(t)\eta_{ih}^{\beta}} & j\in allowed_k \\ 0 & otherwise\end{cases}$$

α is the factor signing the inspiration of information and $beta$ is the factor signing the inspiration of expectation.

<u>Step 3</u> -Update of the pheromone

If the ant a_k move from subinterval I_i to I_j according to the probility, it will leave the pheromone in the subinterval I_j, we note the quantity of pheromone it leaves as $\Delta\iota_j^{(k)}(t)$, so we have $\Delta\iota_j^{(k)}(t)=C_1(f(x_i)-f(x_j))$, the greater the value of $f(x_i)-f(x_j)$, the more pheromone the ant a_k will leave, the more probable the subinterval I_j will appeal to other ants. Each ant who moves to I_j will all leave the pheromone, in this case, we suppose that there are q ants moving to I_j at the t[th] iteration, so the total pheromone they leave is $\Delta\iota_j(t)=\sum_{p=1}^{q}\Delta\iota_j^{(j_p)}(t)$. After all the ants finish this iteration of movement, we update the pheromone as follows: $\iota_j(t+1)=(1-\rho)*\iota_j(t)+\Delta\iota_j(t)$. In which, ρ is the coefficient signing the volatilisation of pheromone and $1-\rho$ is the coefficient signing the remainder of pheromone.

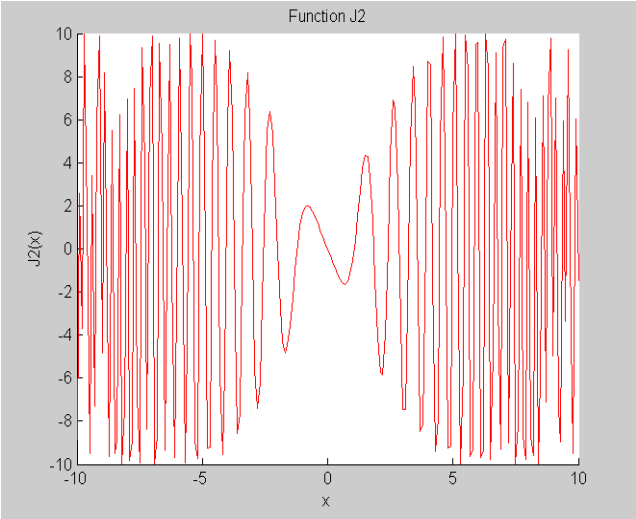
This mechanism is viey reasonable and effective beacause if many ants assemble to one subinterval which means the value of function at this point is very small, the pheromone in this subinterval will be very much and the great quantity of pheromone will appeal to more ants to come, we will finally find this point, it may be the minimum. Otherwise, the subinterval with great function value will be forgotten because less ants pass it and its pheromone will volatilize, thus it will not appeal to the ants to come.

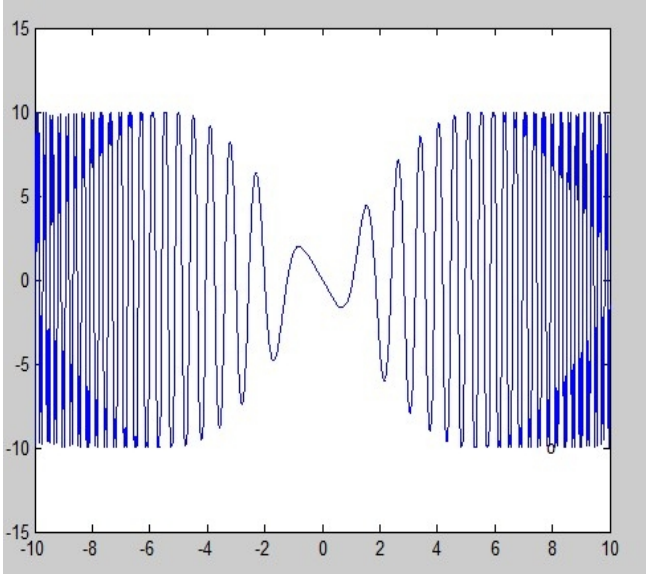
<u>Step 4 -Narrow the research space</u>

After some iterations, when all the ants stop moving, the distribution of intervals will have the feature as follows: the subintervals which contan the ants will must contain the minima; the subintervals which have no ant will impossibly have the minima. We take out these subintervals and refine them: at the next iteration we make the ants search for the minima only in these subintervals. After many iterations, the scope of searching will be smaller and smaller and when the subinterval refined is enough small, all the ants will stop nearby the minima.

3.3 Applications and results

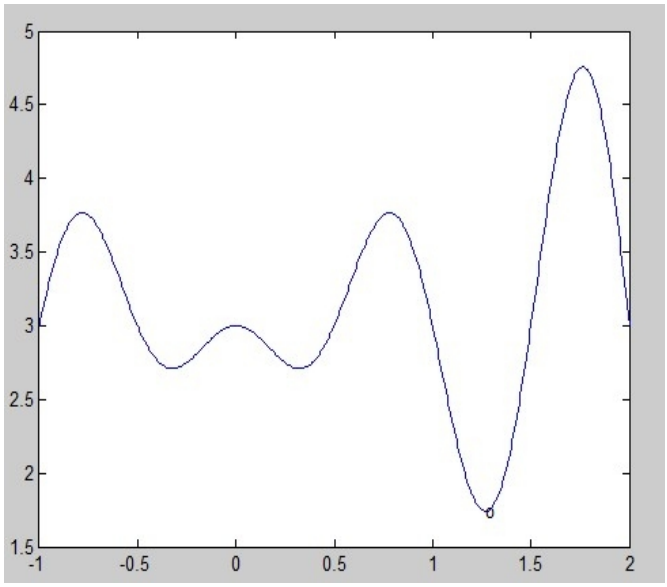
Function $J_2(x) = 10\sin(0{,}3\,x\sin(1{,}3\,x^2 + 0{,}00001\,x^4 + 0{,}2\,x + 80))$





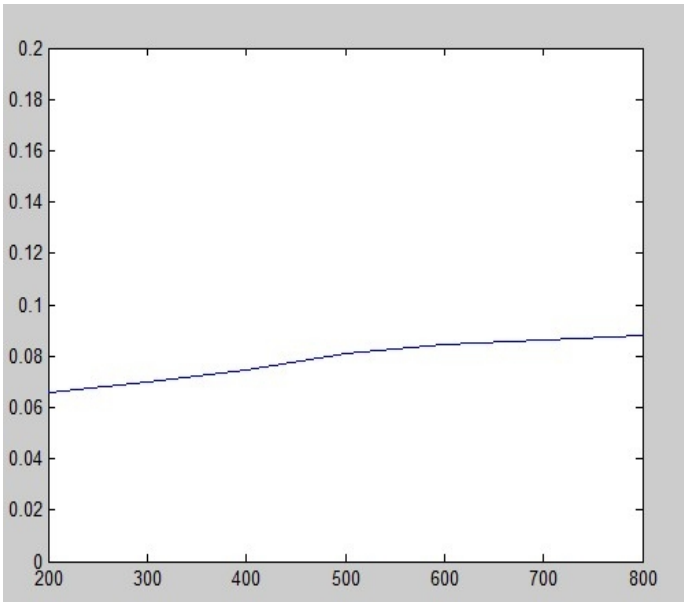
With the ant colony algorithm, we find the global minimum is 7.8130 (the point in cercle) with the error = 0.0001 in 8 iterations.

Function $J_{1,2}(x)=3-x\sin(2\pi x)$



With the ant colony algorithm, We find the global minimum is 1.27 (the point in cercle) with the error = 0.0001 in 7 iterations.

Graph2: The relation between compute time and the number of ants:



IV.Particle swarm optimisation algorithm

1.History

This algorithm has been inspired by the living world. It's based on a model developed by Craig Reynolds at the end of the 80's,enabling to simulate the movement of a group birds or fishes. An other source of inspiration ,defended by some people,belong to social-psychology.
This optimization method is based on mutual collaboration of agents.It's similar to ant colony algorithms,which also rely on a auto-organization concept.This idea supports a group of unintelligent individuals can have a complex global organization.That means each agent moves according to its best past perfomance and the best perfomance of its neighbourhood.

2.Principle,algorithm and choice of parameters

We have N points p_i randomly distributed by an uniform probability law over a part of the space given by a Sobol sequence.

The goal of the algorithm is to assemble these points, that we called henceforth particles, as much as possible to the global minimum of the function that we seek to optimize. The set of particles is called swarm.

For each particle p_i , we define its neighbourhood V_i formed by the set of particles p_j which are "close" to p_i .The notion of closeness can be considered from several different ways.Each of them creates a particular particules 's topology .

Examples of topology :

- In star : each particle is linked to all the others.
- In radius : Particles are connected to an only one central particle.

There are others kinds of topologies .

Algorithm :

The bases of this algorithm are the following :
-Each particle of the swarm have a memory which retain the best solution of the optimization problem it has achieved so far.
-Each particle is enable to communicate with the particles of its neighbourhood.
-Each particle move in the space by following a trend which depends on its intention to go to the best solution it has found (noted $xpBest_i$) and that to want to go to the optimal solution of his neighbourhood.

We intend to solve the following optimisation problem : $\begin{cases} Trouver\ \tilde{x} \in \Omega \\ f(\tilde{x}) = min_{x \in \Omega} f(x) \\ \tilde{x} = argmin_{x \in \Omega} f(x) \end{cases}$.

Step 1 : Create a set of points(particles) uniformly distributed over Ω .

Step 2: Evaluate the function f in each position x_i corresponding to particle p_i .

Step 3 : For each particle,if the current position x_i is better than the best solution known by, ie, if $f(x_i(t)) < f(xpBest_i(t-1))$, then the value of the best solution known by p_i is updated as follows : $xpBest_i(t) = x_i(t)$.

Step 4 : Determine for each particle the position of the best solution obtained by the neighbourhood $xvBest_i(t-1)$ .if the current position is better, ie, if $f(x_i(t)) < f(xvBest_i(t-1))$,then the value of the best solution known by the neighbourhood is updated as folows :
$xpBest_i(t) = x_i(t)$

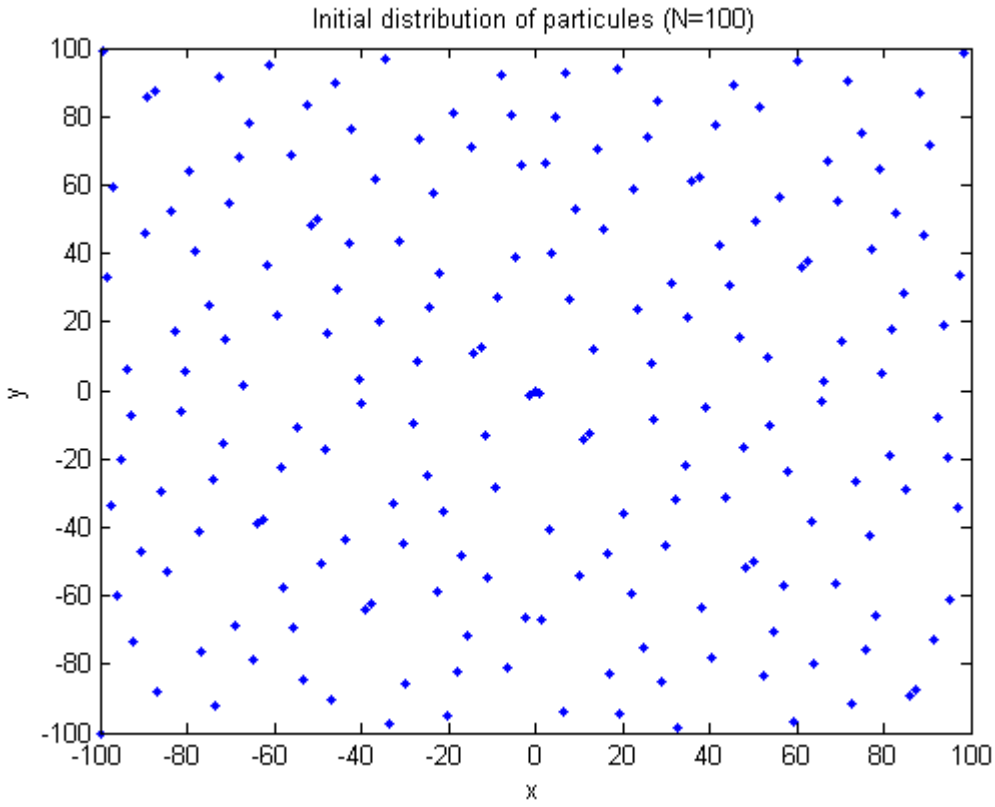
Step 5 : The moving of each particle is computed taking into account a conseravative trend (follow its own solution) and a trend to follow the neighbourhood's solution. The moving is called velocity of the particle.It's defined by :
$v_i(t+1) = \varphi\, v_i(t) + c_1 r_1(t)(xpBest_i(t) - x_i(t)) + c_2 r_2(xvBest_i(t) - x_i(t))$
with $c_1 \geq 0$, $c_2 \geq 0$ et $c_1 + c_2 \leq 4$ and φ called inertia factor.

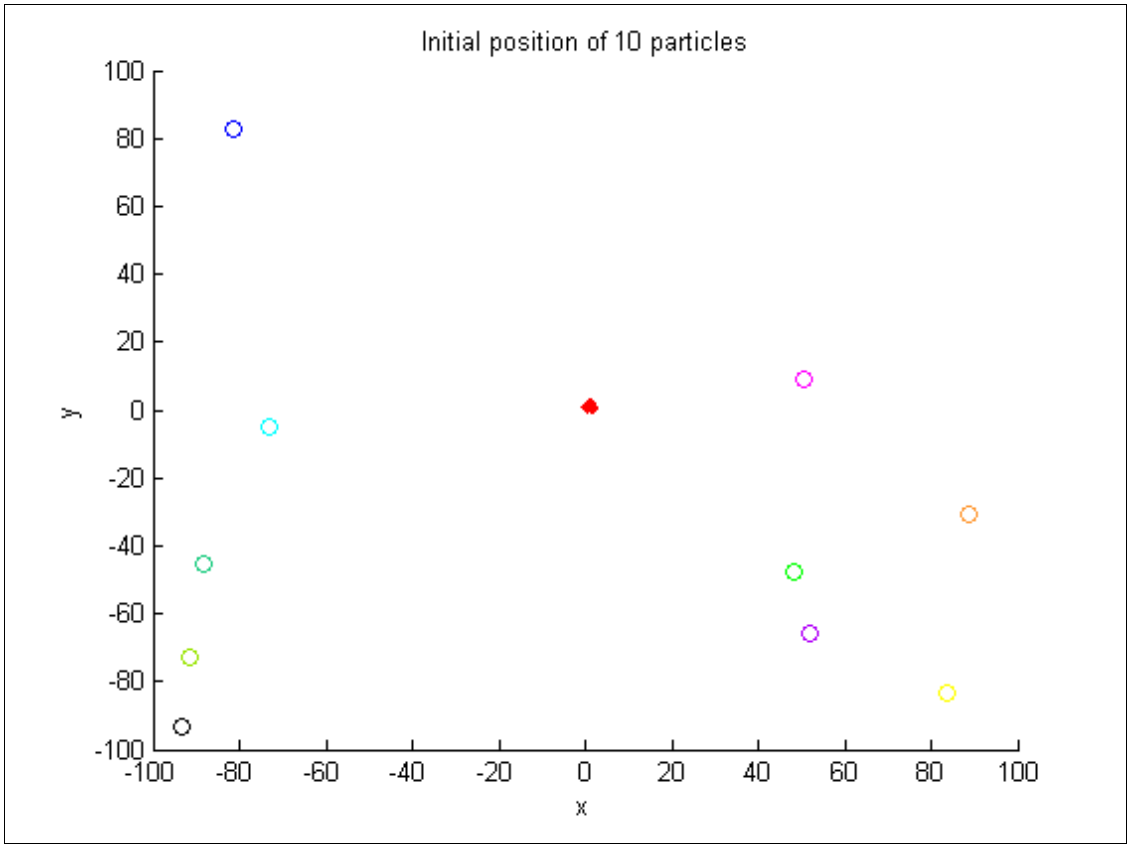
r_1 et r_2 are random variables which follow a uniform law over $[0\,;1]$.

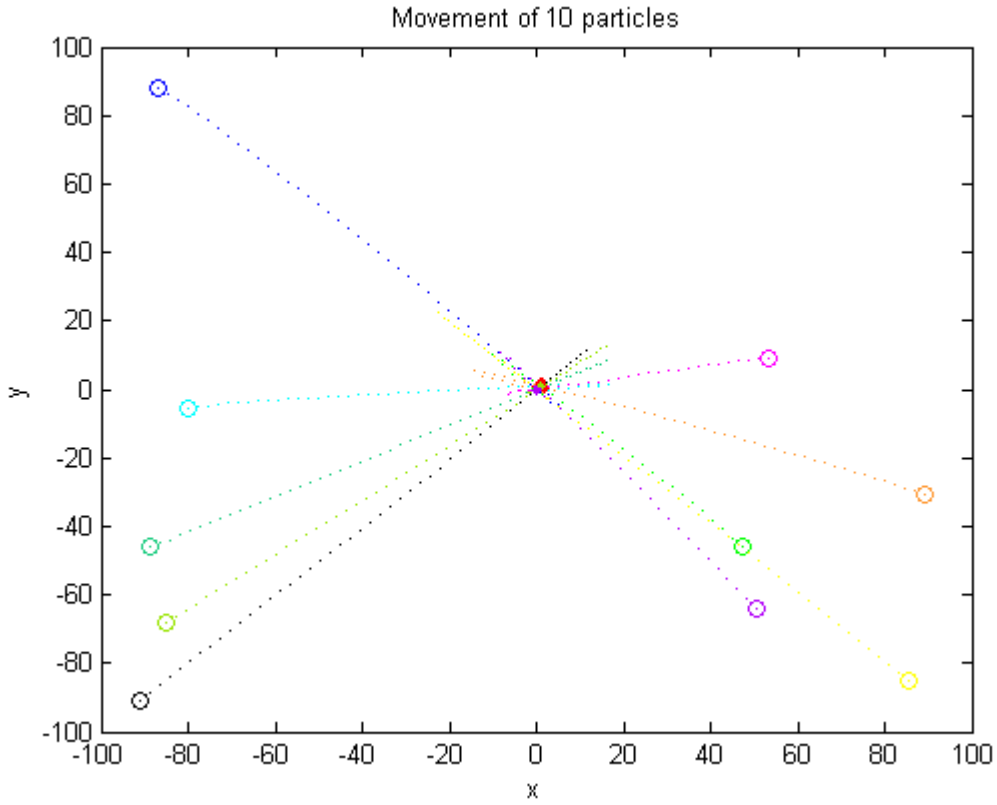
Step 6 :
The particles p_i are moved to their new position by : $x_i(t+1) = x_i(t) + v_i(t+1)$

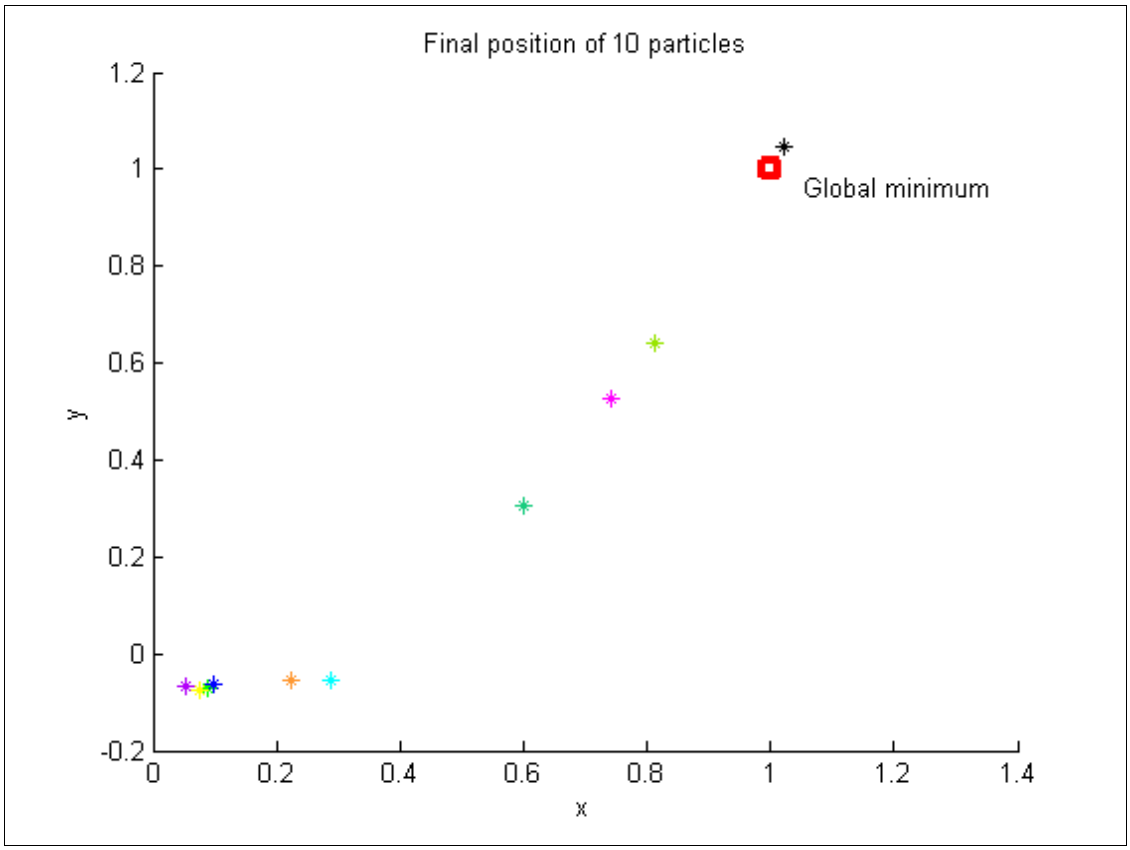
Step 7 :
Go back to step 2 until one of stopping criteria are satisfied.

The following figures give an idea of the particle swarm optimization algorithm with a 2D-function: $J(x, y) = 10(x^2 - y)^2 + (x-1)^2$ Rosenbrock's " Banana function" (see page 15).


Initial distribution of particules (N=100)
100
80
60
40
20
0
-20
-40
-60
-80
-100
y
-100 -80 -60 -40 -20 0 20 40 60 80 100
x


Initial position of 10 particles
100
80
60
40
20
0
-20
-40
-60
-80
-100
y
-100 -80 -60 -40 -20 0 20 40 60 80 100
x


Movement of 10 particles
100
80
60
40
20
0
-20
-40
-60
-80
-100
y
-100
-80
-60
-40
-20
0
20
40
60
80
100
x


Final position of 10 particles
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
y
Global minimum
0
0.2
0.4
0.6
0.8
1
1.2
1.4
x

Choice of parameters

The particule swarm optimization algorithm depends on some parameters which,according to the problem, must be chosen carefully. It's number of particles N, topology and the size of neighbourhood, the inertia factor which impose a maximal velocity,the coefficients c_1 and c_2 . The set of these parameters determine the convergence of the method and above all his speed of convergence.

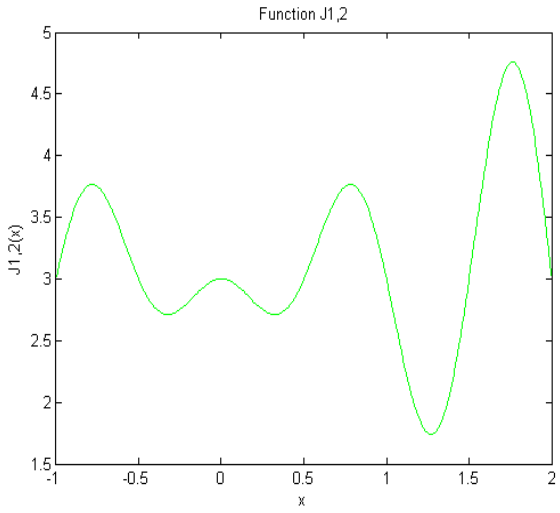
The more particules there are, the faster a global minimum will be approached . The topology is also very important.For example,topology in star guarantee to approach all particles to global minimum ; topology in star would permit to approch central particule to global minimum.

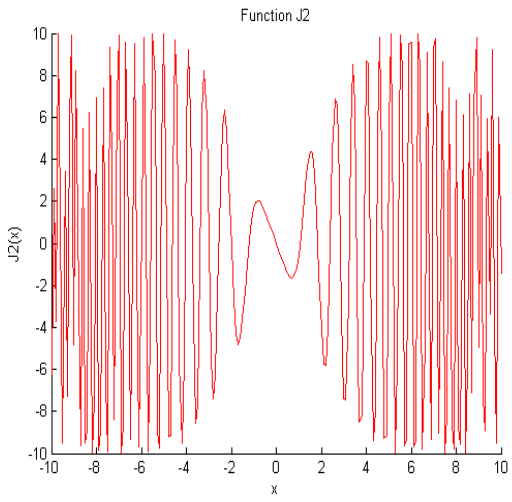
The inertia factor represents the ability to explore of each particule. A great value of φ (> 1) is synonym of a large range of motion, therefore of global exploration. At the opposite a small value of φ (< 1) is synonym of a small range of motion, and therefore of local exploration.

Coefficients c_1 et c_2 characterize respectively conservative trend and trend to follow the neighbourhood. If the first is greater than the other,a particle would try to approach its best solution more than that of his neighbourhood and inversely. The two parameters must be chosen also relying on the size of space to explore. For a continuous fonction which often varies ,thus which has numerous extrema, they must be taken enough small.

3.Applications and results

In a space of dimension 1





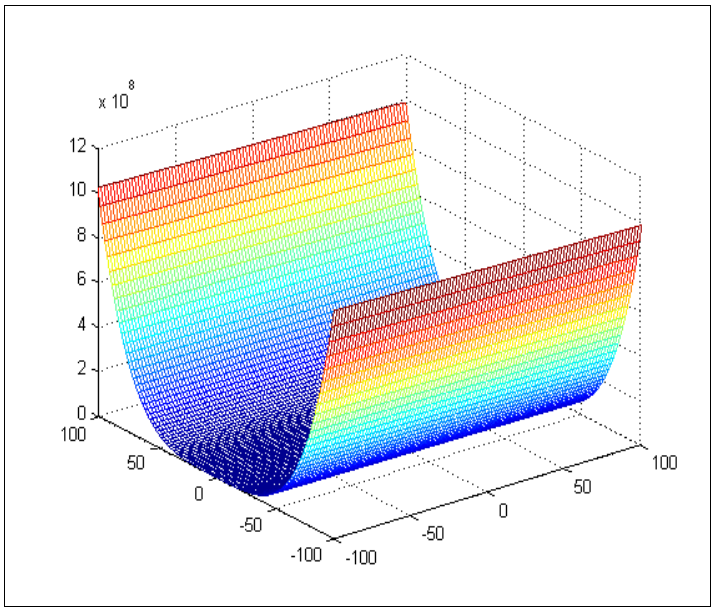
Function J1,2: $J_{1,\omega}(x)=3-x\sin(\omega\pi x)$
The algorithm converges in 270 iterations ,the computing time is 0.964703 seconds and the minimum global found is x=1.2698 .

Function J2: $J_2(x)=10\sin(0{,}3\,x\sin(1{,}3x^2+0{,}00001\,x^4+0{,}2\,x+80))$

The algorithm converges in 2500 iterations ,the computing time is 8.182380 seconds and the minimum global found is x=-7.8122.

In a space of dimension 2

Function: $J(x,y)=10(x^2-y)^2+(x-1)^2$
Graph: Rosenbrock's " Banana function"



The algorithm converges in 302 iterations ,the computing time is 0.51146 and the minimum global found is [1.0011 , 1.0022] .

In a space of dimension 3
Function J: $J(x,y,z)=(x-1)^2+(y-1)^4+(z+11)^2-5;$
The algorithm converges in 302 iterations ,the computing time is 0.511464 seconds and the minimum global found is x=[-0.6998 , 2.3145 , -10.0694]

In a space of dimension n

Function J: $J_n(x)=\frac{1}{2}x^T\cdot A_n\cdot x-b_n\cdot x$ with $A_n=\begin{pmatrix} 4 & -2 & 0 & \dots & 0 \\ -2 & 4 & -2 & \dots & 0 \\ 0 & -2 & 4 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -2 & 4 \end{pmatrix}$ a (n,n)-matrix and

$b_n=[1,1\dots,1]$ a n-size vector.

With n=3, the algorithm converges in 860 iterations ,the computing time is 10.661960 seconds and the minimum global found is x=[0.7498 , 0.9995 , 0.7491] .

With n=10, the algorithm converges in 874 iterations,the computing time is 11.496196 seconds and the minimum global found is:
x=[2.8426 , 4.3783 , 5.1186 , 5.6579 , 6.1691 ,6.3097 ,5.9203 ,5.3953 ,5.1275 ,3.2622]

With n=100, the algorithm converges in 5312 iterations ,the computing time is 155.197859 seconds and the minimum global found is a vector of 100 components.

V. Comparison of algorithms

1.Calculation time

Function J1,2:

Optimisation algorithm	**Calculation time**
Simulated annealing	0.015544 seconds
Ant Colony	0.063744 seconds
Particle swarm	0.964703 seconds

Function J2:

Optimisation algorithm	**Calculation time**
Simulated annealing	0.006461 seconds
Ant Colony	0.065763 seconds
Particle swarm	8.182380 seconds

Comments:The simulating annealing is the fastest.

2.Number of iterations

Function J1,2:

Optimisation algorithm	**Number of iterations**
Simulated annealing	3000
Ant Colony	6
Particle swarm	270

Function J2:

Optimisation algorithm	**Number of iterations**
Simulated annealing	3000
Ant Colony	7
Particle swarm	2500

Comments: The number of iterations of ant colony algorithm is the least of the three.The others need very more iterations.

3. Precision

Function J1,2:

Optimisation algorithm	**Error (Precision)**
Simulated annealing	10^-3
Ant Colony	10^-2
Particle swarm	10^-3

Function J2:

Optimisation algorithm	**Error(Precision)**
Simulated annealing	10
Ant Colony	2.10^-1
Particle swarm	10^-3

Comments: Ant colony and particle swarm gives a lower error than the simulated annealing.

4.Specifities

The simulated annnealing is faster than the two other but less precise and if it converges,it can be need many iterations to converge to a global minimum.

The ant colony algorithm is faster than the particle swarm but it is not simple to program for function which have more than 2 variables. In the case of one-dimension function it's as precise as the particle swarm.

The particule swarm has the main advantage to be usable for a function of any dimension and its precision can be chosen or controlled.

Conclusion

To sum up, we have tested three algorihms for continuous optimization: find a global minimum for real, continuous, differentiable and non-linear functions.Each algorithm has its own advantage, however since they all aim to find the global optimum, so they take a little time than classical algorithms giving local minimum.