FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION

OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

on the practical task No. 4

“Algorithms for unconstrained nonlinear optimization. Stochastic and metaheuristic algorithms”

Performed by

Cheng-Yuan and Ma

Academic group

J4133c

Petr Chunaev *St.*

Petersburg 2022

Task# 4

Language : python

Goal : The use of stochastic and metaheuristic algorithms (Simulated Annealing, Differential Evolution, Particle Swarm Optimization) in the tasks of unconstrained nonlinear optimization and the experimental comparison of them with Nelder-Mead and Levenberg-Marquardt algorithms

Problem : Generate the noisy data (x , y_k), where k = 0, ... ,1000, according to the rule:

*X = 3*k / 1000*

Y = { -100 + δ , f(x) < -100

f(x) + δ , -100 < ,f(x) < 100

100+δ , ,f(x) > 100}

where $\delta_k \sim N(0,1)$ *are values of a random variable with standard normal*

distribution. Approximate the data by the rational function

$F(x,a,b,c,d)$ = ax + b / x^2 + cx + d

by means of least squares through the numerical minimization of the following function:

$D(a,b,c,d) = \sum (F(x_k,a,b,c,d) - y_k)^2$.

I.

Theory:

Levenberg- Marquardt algorithm :

This is one algorithm which combine Newton method and gradient descent .

It applicate two method depends on situation , and alter lambda , a learning rate which can adjust proportion of two methods , depending on the steepness of slope

Hessian matrix = H(xk)
Gradient matrix = f(xk)
xk+1 = xk - (H(xk)+ lambda*Identity matrix)^-1 X f(xk)

Simulated Annealing :

(Pseudo code)

```
T_MAX = 1000 # initial temperature

T_MIN = 10 # Set end temperature

T = T_MAX # set the current temperature


markov_step = 100 # Set the number of times the Markov chain learns

current_state = generate_init_state() # Randomly generate initial state

current_energy = energy_function(current_state, T) # generate current energy


while T has not yet reached T_MIN

    for i=1:markov_step

        new_state = generate_new_state(current_state) # Generate a new state based on the current state

        new_energy = energy_function(new_state, T) # Calculate the energy of the new state

        alpha = acceptance(new_energy, current_energy, T) # Calculate an acceptance probability value via the function

        if rand() <= alpha

            # accept new state

        else

            # reject new state

        end

        # cool down

    end

end
```

Particle Swarm Optimization :

The particle swarm optimization algorithm mainly has three processes, and the completion of these three processes is called an iteration.

Assign the starting position, and record the individual best solution and the group best solution

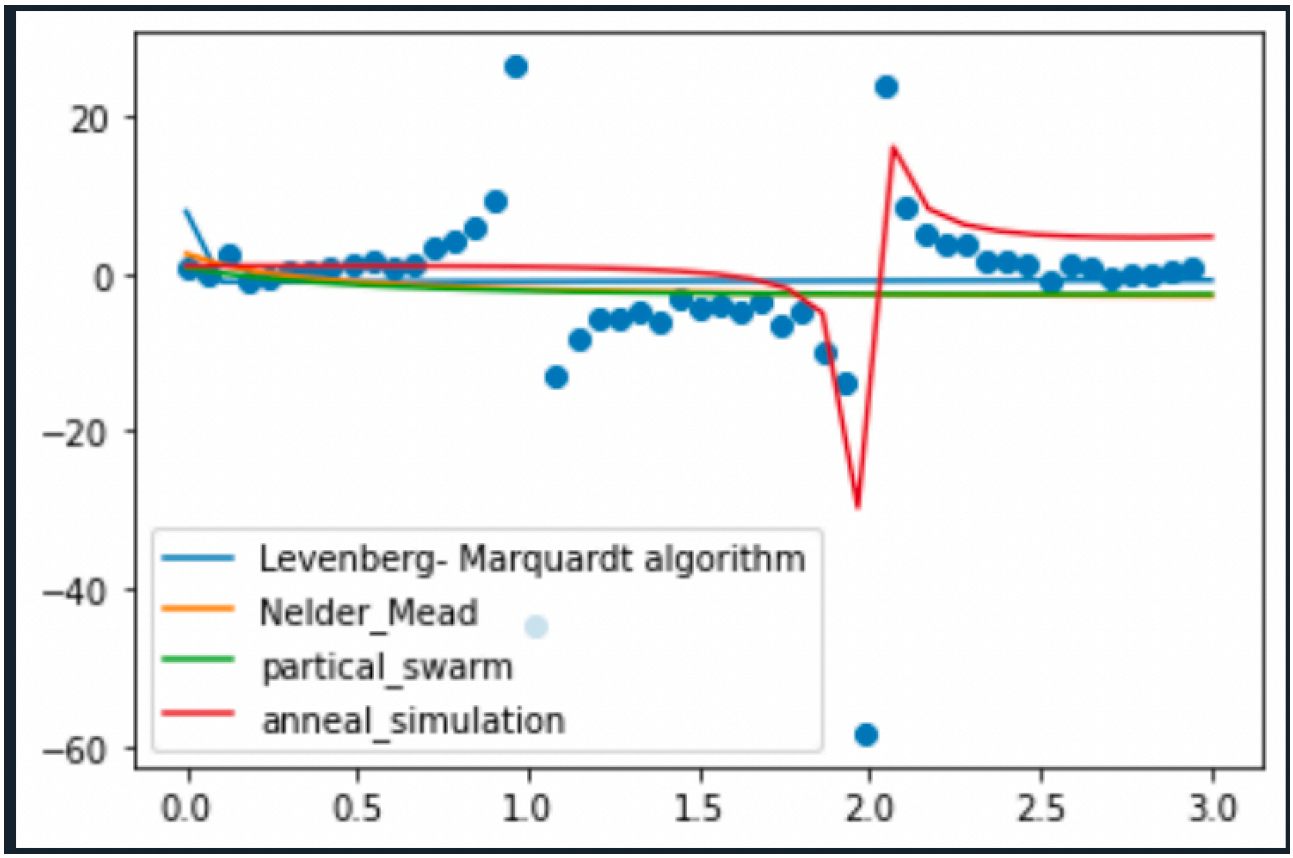
Calculate the acceleration vector to move the migratory flock to a new location.

Update individual best solution and global best solution

Nelder-Mead : is a numerical method used to find the minimum or maximum of an objective function by reflection, expansion, contraction and shrink coefficients in a multidimensional space

Result :

Instead of setting range from 0 to 1000 , I chose node every 20 from 0 to 1000 , due to long period of process




```
_________________
Levenberg- Marquardt algorithm :initial_node=[-10, -10, 7, 0] , iteration=32 , cost=152.620140520017 , evaluation of function=36
Nelder_Mead  :initial_node=[10, 10, -9, 0] , iteration=486 , cost=150.4815758209012  ,evalutaion of function=800
partical_swarm  :initial_node=[5, 5, 5, 5] , iteration=30 , cost=150.2433869525066  ,evalutaion of function=450
Simulated Annealing  : iteration=228,cost=1.00,evaluation of function=22801.000000
```

Conclusion :

In this task, I only used node every 20 node , from 0 to 1001 , instead of every node from 0 to 1001 , somehow it cost a super long time , which it did not show any output in first 30 minutes .

I tried to use random initial points at first, then I realized that it's really hard to find optimal points in multidimensional graphs, most of them get stuck after some iterations, and give me very good The output is similar to the initial point. I started experimenting with points close to the optimal point. As the fitted lines shown in the figure, we can see that most of them can converge on the same line, even if the starting point of the line is biased, but in the end they all try to fit on a line. The simulated annealing results are slightly different and generally the same as the other algorithms, but the most impressive part is that it deviates when some nodes deviate significantly from the majority.

Appendix

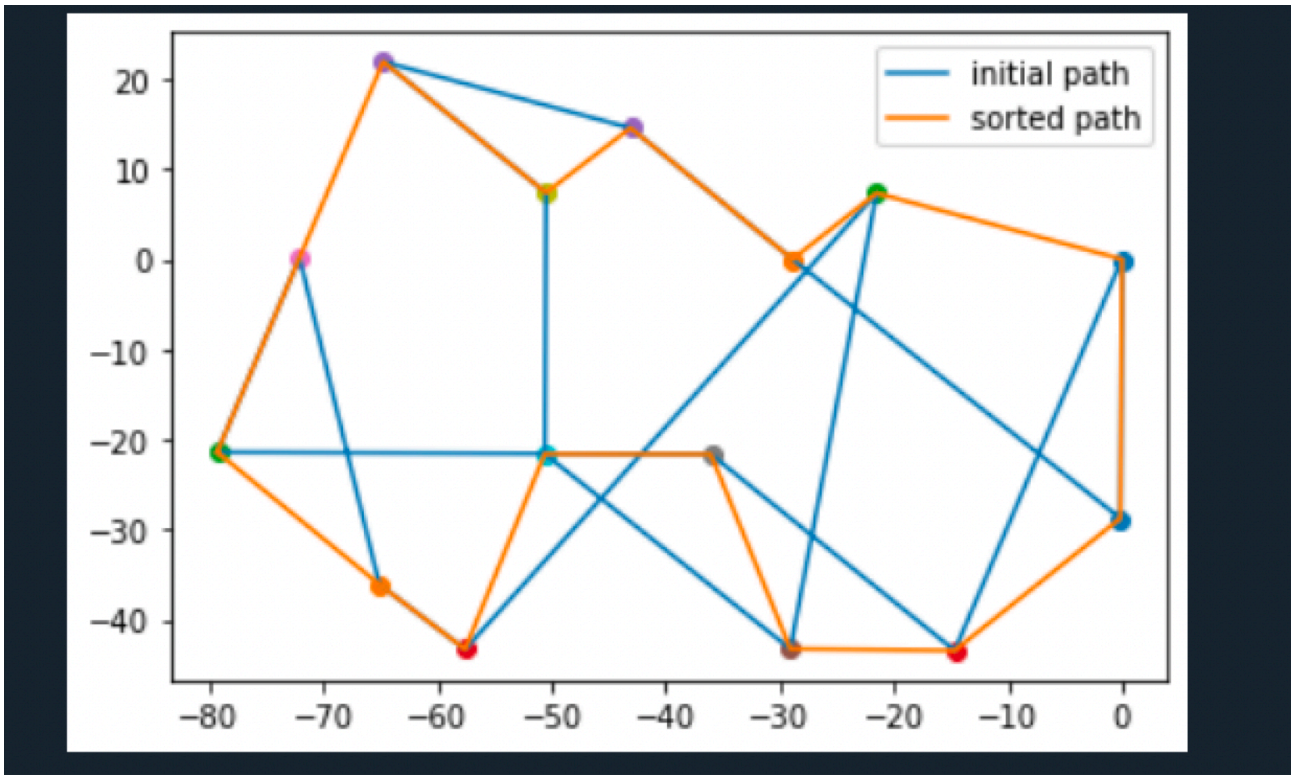
https://github.com/MaChengYuan/task4.git

II.

Problem :

Choose at least 15 cities in the world having land transport connections between them. Calculate the distance matrix for them and then apply the Simulated Annealing method to solve the corresponding Travelling Salesman Problem. Visualize the results at the first and the last iteration.

Result :




```
-- random solution --
Chicago -> Boston -> Minneapolis -> New York -> Houston -> Los Angeles -> russia -> Denver -> San Francisco -> St. Louis -> Dallas -> Salt Lake City -> taiwan
-> Phoenix -> Seattle -> Chicago

Total distance: 495.2783441492317 miles

iteration : 14
-- Simulated annealing solution --
Chicago -> Chicago -> Phoenix -> taiwan -> St. Louis -> Boston -> Dallas -> Minneapolis -> Houston -> New York -> Salt Lake City -> Los Angeles -> russia ->
San Francisco -> Denver -> Seattle -> Chicago -> Chicago

Total distance: 284.3810904079913 miles
```

Conclusion :

As we can see , it gets to conclusion very fast , it only takes 14

Iteration and successfully reduced total distance by 200 miles .

It is quite magnificent accomplishment if it can be used on technology .

Appendix

https://github.com/MaChengYuan/task4.git