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
Zakhar Pinaev
J4132c
Accepted by
Dr Petr Chunaev

St. Petersburg
2021

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

Problems and methods

I. Approximate the generated data by rational functions by means of least squares through the numerical minimization (with precision $\varepsilon = 0.001$) of the given function. To solve the minimization problem, use Nelder-Mead algorithm, Levenberg-Marquardt algorithm and at least two of the methods among Simulated Annealing, Differential Evolution and Particle Swarm Optimization. If necessary, set the initial approximations and other parameters of the methods. Use $\varepsilon = 0.001$ as the precision; at most 1000 iterations are allowed. Visualize the data and the approximants obtained in a single plot. Analyze and compare the results obtained (in terms of number of iterations, precision, number of function evaluations, etc.).

II. 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.

Brief theoretical part

When solving a specific optimization problem, the researcher must first choose a mathematical method that would lead to final results with the least computational costs or make it possible to obtain the largest amount of information about the desired solution. The choice of one method or another is largely determined by the formulation of the optimal problem, as well as the mathematical model of the optimization object used.

Common to nonlinear programming methods is that they are used to solve problems with nonlinear optimality criteria. All nonlinear programming methods are search-type numerical methods. Their essence lies in the definition of a set of independent variables that give the greatest increment of the optimized function.

Deterministic heuristic (greedy) methods are based on the idea of locally optimal choices at each step. In many optimization problems, the efficiency of greedy algorithms is highly dependent on the conditions of a particular problem. Stochastic methods also do not guarantee an exact solution, but they, as a rule, allow finding solutions that are close enough for practical use in a reasonable time. At the same time, the stochastic nature of most of the methods makes their application a non-trivial task, since for each algorithmic implementation and for each class of optimization problems, the efficiency, speed, convergence, the influence of the problem conditions and algorithm parameters require careful research.The algorithms of considered below, like evolutionary algorithms and the algorithm for simulating annealing have one basis, namely, finite Markov chains.

Results

I. In the task, it was necessary to generate data in a certain way and approximate it by rational function by means of least squares through the numerical minimization of the given function. To

solve the minimization problem, were used Nelder-Mead algorithm, Levenberg-Marquardt algorithm, Simulated Annealing and Differential Evolution algorithm.

For the corresponding generated data, Figure 1 shows approximation by rational function using Nelder-Mead algorithm, Levenberg-Marquardt algorithm, Simulated Annealing and Differential Evolution algorithm.

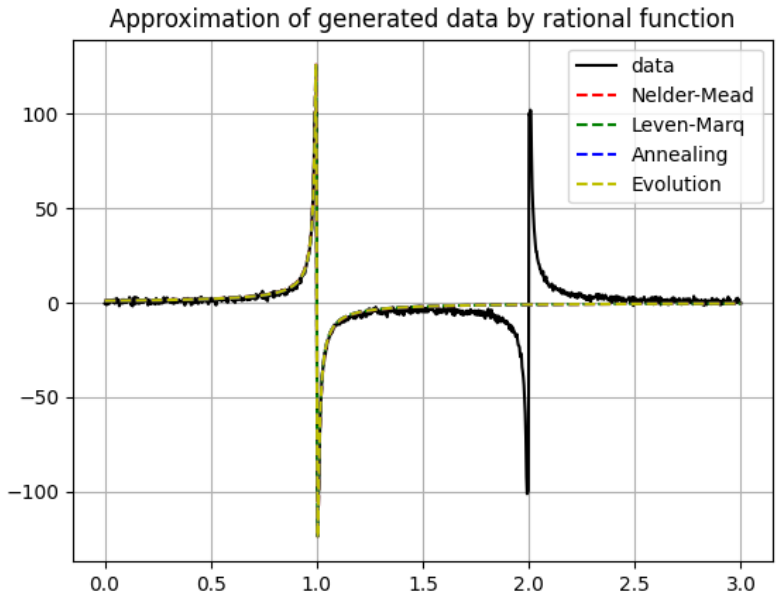

Figure 1 – rational approximating function using Nelder-Mead algorithm (red dashed), Levenberg-Marquardt algorithm (green dashed), Simulated Annealing (blue dashed), Differential Evolution algorithm (yellow dashed), data is shown via black line

As can be seen from Figure 1, all methods gave extremely close results, therefore, for a more detailed comparison, table 1 shows the detailed results for calculating the approximation functions by different methods.

Table 1 – Detailed results of calculating rational approximation of function

method	result (minimization function)	number of iterations	*f*-calculations
Nelder-Mead	137986.815	137	253
Levenberg-Marquardt	137986.684	13	26
Simulated Annealing	137986.964	970	1940
Differential Evolution algorithm	137986.964	647	1295

It can be seen from Table 1, that, as a result, all minimization methods gave approximately the same values of the minimization function. As expected, stochastic methods required more

iterations and calculations of functions to find a solution with a given accuracy, because they are more or less based on randomness. The Levenberg-Marquardt algorithm proved to be the best, and in the framework of stochastic methods, the differential evolution algorithm turned out to be the best.

II. In the second task, the solution of the traveling salesman problem was considered by the simulated annealing method. For this, the coordinates of 128 cities in North America were taken, between which there is a ground transport connection. Further, the distance matrix was calculated for the selected cities, and then the simulated annealing method was applied to solve the corresponding traveling salesman problem. Figures 2-3 show the results of the algorithm operation at the first and last iterations, respectively.

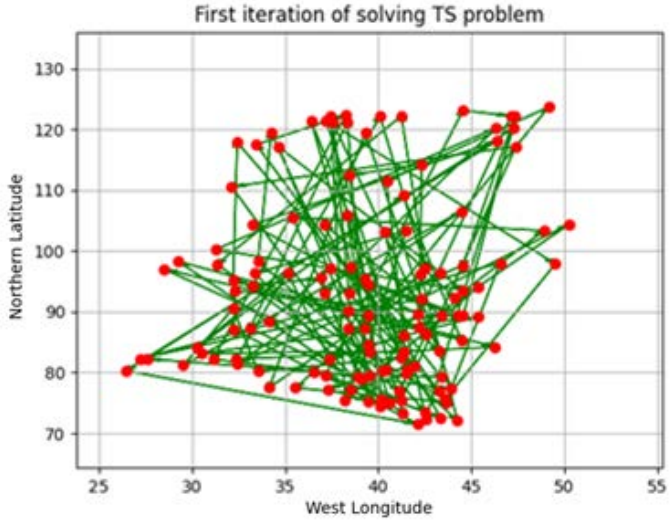

Figure 2 – Route visualization obtained at the first iteration of solving the traveling salesman problem

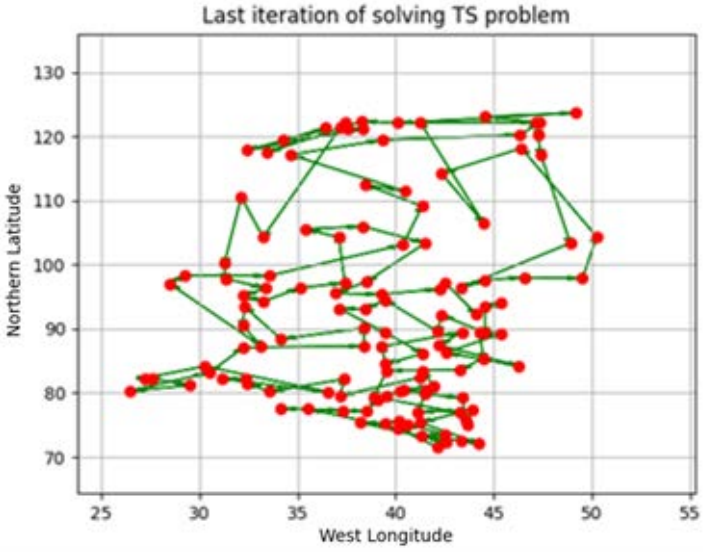

Figure 3 – Route visualization obtained at the last iteration of solving the traveling salesman problem

The route at the first iteration (Figure 2) was determined randomly and its length was 2701.635 km. In turn, the length of the route built by the annealing simulation algorithm was 493.352 km. Thus, the route compiled by the algorithm at the last iteration turned out to be 81.74% shorter than the original route at the first iteration. Figure 4 shows a graph of the route length change at each iteration of the simulated annealing algorithm.

Figure 4 – The graph of the change in the length of the route depending on the iteration of the simulated annealing algorithm

After that, for the experiment, it was decided to use the result of the greedy algorithm (the nearest neighbor algorithm) as the starting route (the route at the first iteration). Figures 5, 6 show the route at the first and last iteration of this solution.

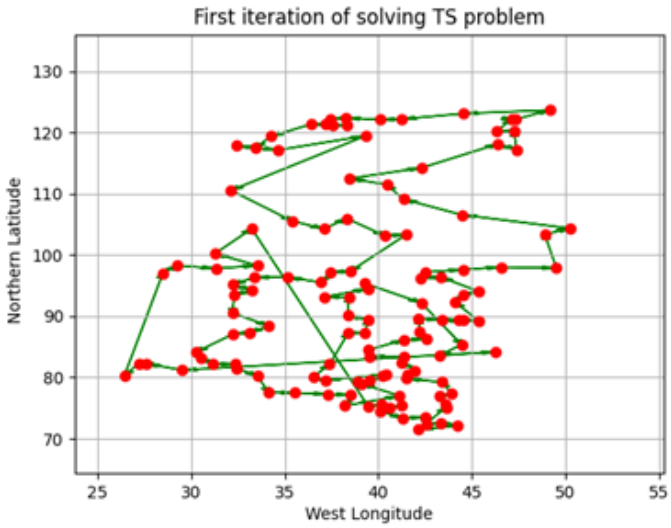

Figure 5 – Route visualization obtained at the first iteration of solving the traveling salesman problem with using nearest neighbor algorithm as first iteration

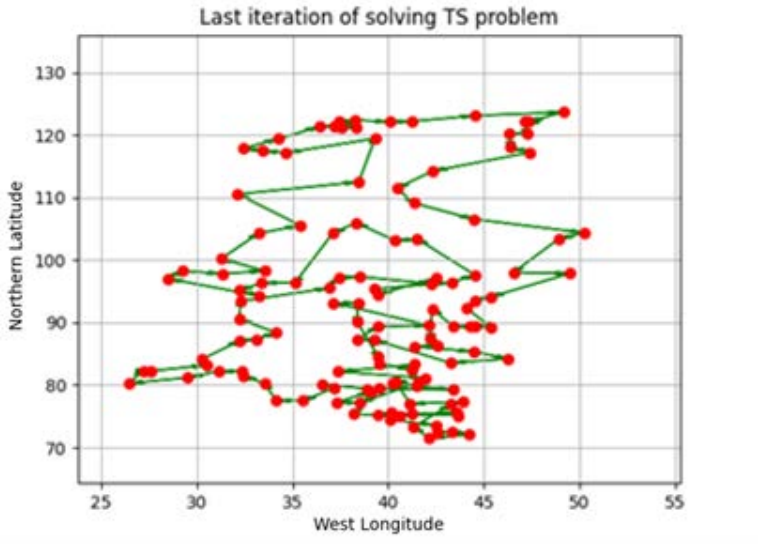

Figure 6 – Route visualization obtained at the last iteration of solving the traveling salesman problem with using nearest neighbor algorithm as first iteration

The length of the original route, built using the nearest neighbor algorithm, was 329.99 km. In turn, the length of the path obtained at the last iteration of the annealing simulation algorithm was 323.80 km. Thus, the simulated annealing algorithm reduced the path built by the nearest neighbor algorithm by 1.87%. Figure 7 shows a graph of the dependence of the route length on the number of iterations of the annealing simulation algorithm.

Figure 7 – The graph of the change in the length of the route depending on the iteration of the simulated annealing algorithm with using nearest neighbor algorithm as first iteration

Conclusion

As a result of the work, stochastic and metaheuristic methods were investigated in problems of unconstrained nonlinear optimization. In the first task, stochastic methods were used to minimize the function in the problem of approximating the generated data. Among the stochastic methods considered, the differential evolution method turned out to be more effective, however, when

compared with other methods, the Levenberg-Marquardt algorithm turned out to be more effective in the framework of this problem.

In the second task, the solution of the traveling salesman problem was considered using the simulated annealing algorithm. It turned out that the simulated annealing method creates shorter routes not only for a randomly assigned starting route, but also in the case when the starting route is obtained by a greedy algorithm. Thus, to get the best route in the most efficient way, these algorithms can be combined with using the greedy algorithm as a first-iteration route.

Appendix

```
import math
import numpy as np
import random
import matplotlib.pyplot as plt
from scipy.optimize import least_squares
from scipy.optimize import dual_annealing
from scipy.optimize import differential_evolution
from scipy.optimize import minimize
from anneal import SimAnneal


def f(x):
    return 1 / (x ** 2 - 3 * x + 2)


def generate_data(N):
    k = range(0, N+1)
    x = [0] * (N+1)
    y = [0] * (N+1)
    sigma = np.random.standard_normal(N+1)
    for i in range(N+1):
        x[i] = (3 * k[i]) / 1000
        fx = f(x[i])
        if fx < -100:
            y[i] = -100 + sigma[i]
        elif fx >= -100 and fx <= 100:
            y[i] = fx + sigma[i]
        elif fx > 100:
            y[i] = 100 + sigma[i]

    return x, y


def approx_ratio(x, a, b, c, d):
    return (a * x + b) / (x ** 2 + c * x + d)


def num_minim(x, y, a, b, c, d):
    D = 0
    for k in range(len(y)):
        D += (approx_ratio(x[k], a, b, c, d) - y[k]) ** 2
    return D


def num_minimize_vec(point, x, y):
    a, b, c, d = point
    D = 0
    for k in range(len(y)):
        D += (approx_ratio(x[k], a, b, c, d) - y[k]) ** 2
```

```
    return D


def minim(x0, x, y):
    a, b, c, d = x0
    D = 0
    res = [(a * xx + b) / (xx ** 2 + c * xx + d) - yy for xx, yy in zip(x,
y)]
    return res


def nelder_mead(x, y, prec=1e-3):
    x0 = np.array([-1.0, 1.0, -2.0, 1.0])

    res = minimize(num_minimize_vec, x0, args=(x, y), tol=prec,
method='Nelder-Mead')

    return res.x[0], res.x[1], res.x[2], res.x[3], res.nfev, res.nit,
num_minim(x, y, res.x[0], res.x[1],

res.x[2], res.x[3])


def leven_marq(x, y, prec=1e-3):
    x0 = np.array([-1.0, 1.0, -2.0, 1.0])

    res = least_squares(minim, x0, args=(x, y), ftol=prec, xtol = prec,
method='lm')

    return res.x[0], res.x[1], res.x[2], res.x[3], res.nfev, res.nfev // 2,
num_minim(x, y, res.x[0], res.x[1],

res.x[2], res.x[3])


def annealing(x, y, prec=1e-3):
    x0 = [-1.0, 1.0, -2.0, 1.0]
    lw = [-10] * 4
    up = [10] * 4

    res = dual_annealing(num_minimize_vec, args=(x, y), x0 = x0,
restart_temp_ratio=prec, bounds=list(zip(lw, up)))

    return res.x[0], res.x[1], res.x[2], res.x[3], res.nfev, res.nfev // 2,
num_minim(x, y, res.x[0], res.x[1],

res.x[2], res.x[3])


def diff_evolution(x, y, prec=1e-3):
    x0 = [-1.0, 1.0, -2.0, 1.0]
    lw = [-10] * 4
    up = [10] * 4

    res = differential_evolution(num_minimize_vec, args=(x, y), x0 = x0,
bounds=list(zip(lw, up)))

    return res.x[0], res.x[1], res.x[2], res.x[3], res.nfev, res.nfev // 2,
num_minim(x, y, res.x[0], res.x[1],

res.x[2], res.x[3])


def read_coords(path):
```

```
    coords = []
    with open(path, "r") as f:
        for line in f.readlines():
            line = [float(x.replace("\n", "")) for x in line.split(" ")]
            #print(line)
            coords.append(line)
    return coords


def main():
    """
    precision = 0.001
    N = 1000
    x, y, = generate_data(N)

    a_1, b_1, c_1, d_1, it_1, fc_1, dd_1 = nelder_mead(x, y, precision)
    a_2, b_2, c_2, d_2, it_2, fc_2, dd_2 = leven_marq(x, y, precision)
    a_3, b_3, c_3, d_3, it_3, fc_3, dd_3 = annealing(x, y, precision)
    a_4, b_4, c_4, d_4, it_4, fc_4, dd_4 = diff_evolution(x, y, precision)

    print(dd_1, a_1, b_1, c_1, d_1, it_1, fc_1)
    print(dd_2, a_2, b_2, c_2, d_2, it_2, fc_2)
    print(dd_3, a_3, b_3, c_3, d_3, it_3, fc_3)
    print(dd_4, a_4, b_4, c_4, d_4, it_4, fc_4)

    y_aprox_1 = [(a_1 * xx + b_1) / (xx ** 2 + c_1 * xx + d_1) for xx in x]
    y_aprox_2 = [(a_2 * xx + b_2) / (xx ** 2 + c_2 * xx + d_2) for xx in x]
    y_aprox_3 = [(a_3 * xx + b_3) / (xx ** 2 + c_3 * xx + d_3) for xx in x]
    y_aprox_4 = [(a_4 * xx + b_4) / (xx ** 2 + c_4 * xx + d_4) for xx in x]

    plt.plot(x, y, 'k-', x, y_aprox_1, 'r--', x, y_aprox_2, 'g--', x,
y_aprox_3, 'b--', x, y_aprox_4, 'y--')
    plt.title('Approximation of generated data by rational function')
    plt.legend(['data', 'Nelder-Mead', 'Leven-Marq', 'Annealing',
'Evolution'], loc='best')
    plt.grid(True)
    plt.show()

    """

    ################################### task 2
###################################
    coords = read_coords("sgb128_coords.txt")  # generate_random_coords(100)
    sa = SimAnneal(coords, stopping_iter=10000)
    sa.anneal()
    sa.visualize_routes()
    sa.visualize_first()
    sa.plot_learning()


if __name__ == "__main__":
    main()
```