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

*Mariia Koroleva*

*J42322c*

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*

**Formulation of the problem**

*Generate the noisy data , where , according to the rule:*

*where are values of a random variable with standard normal distribution. Approximate the data by the rational function*

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

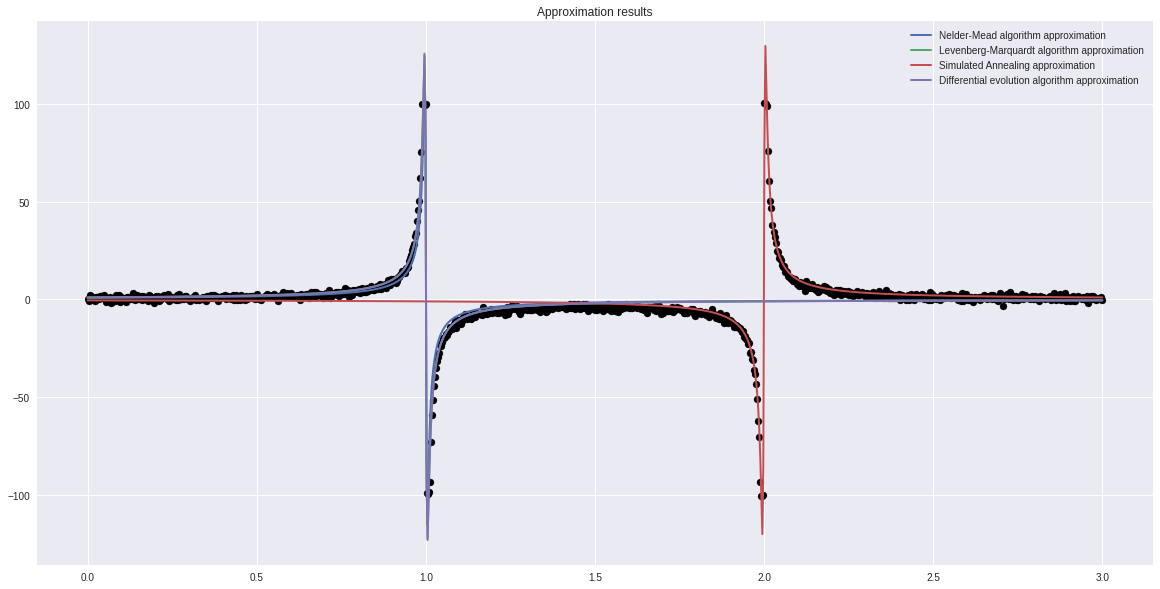
*To solve the minimization problem, use Nelder-Mead algorithm, ﻿Levenberg-Marquardt algorithm and* ***at least two*** *of the methods among Simulated Annealing, Differential Evolution and Particle Swarm Optimization. If necessary, set the initial approximations and other parameters of the methods. Use 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.).*

**Brief theoretical part**

*Simulated Annealing* is a metaheuristic evolutionary algorithm, that solves the optimization problem similar to the process of annealing in metallurgy.

*Differential Evolution* is a metaheuristic algorithm that solves the optimization problem by maintaining a population of agents, i.e. candidate solutions, creating new agents by combining existing ones and further keeping the best one.

**Results**



Note: LM approximation and differential evolution approximation graphs coincide, that is why we cannot see here the green LM’s approximation graph.

**Conclusions**

* There are no significant graphical differences between the approximations. Only Simulated Annealing stands out.
* The accuracy of all algorithms is similar, and the lowest minimum is shown in Levenberg-Marquardt algorithm.
* Judging by such an indicator as the number of iterations, the most efficient algorithm is Differential Evolution and the least efficient is Simulated Annealing
* Judging by such an indicator as the number of function evaluations, the most efficient algorithm is Levenberg-Marquardt algorithm and the least efficient is Simulated Annealing

**Appendix**

*# modules*

import numpy as np

import pandas as pd

from scipy import optimize

import matplotlib.pyplot as plt

eps = 0.001

f = **lambda** x: 1/(x\*\*2 - 3\*x + 2)

k = np.array(list(range(0, 1001)))

X = np.array(list(map(**lambda** x: 3\*x/1000, k)))

**def** y(x):

    if f(x) < - 100:

        return -100 + np.random.normal(0, 1)

    elif f(x) > 100:

        return 100 + np.random.normal(0, 1)

    else:

        return f(x) + np.random.normal(0, 1)

Y= np.array([y(x) for i, x in enumerate(X)])

approx = **lambda** x, a, b, c, d: (a\*x + b)/(x\*\*2 + c\*x + d)

D = **lambda** args: sum([(approx(x, \*args) - y) \*\* 2 for x, y in zip(X, Y)])

results = pd.DataFrame()

algs = ['Nelder-Mead', 'Levenberg-Marquardt','Simulated Annealing', 'Differential Evolution']

results['Algorithm name'] = algs

results.set\_index('Algorithm name', inplace=True)

results.style.set\_caption("Results of approximation")

header\_list = ['Number of iters', 'Function min', 'Number of func calculations']

results = results.reindex(columns = header\_list)

**# Nelder-Mead**

**def** nelder\_mead(f, X, Y):

    result = optimize.minimize(D, np.random.rand(4), method ='nelder-mead', tol=0.001)

    return result

opt\_Nelder = nelder\_mead(approx, X, Y)

num\_of\_iterations, func\_value, func\_count = opt\_Nelder.nit, round(opt\_Nelder.fun, 3), opt\_Nelder.nfev

results.loc[algs[0], :] = [num\_of\_iterations, func\_value, func\_count]

**# Levenberg-Marquardt**

approx\_corrected = **lambda** args, x, y: approx(x, \*args) - y

opt\_LM = optimize.least\_squares(approx\_corrected, np.random.rand(4), args=(X, Y), method='lm')

num\_of\_iterations, func\_value, func\_count = '-', round(D(opt\_LM.x), 3), opt\_LM.nfev

results.loc[algs[1], :] = [num\_of\_iterations, func\_value, func\_count]

%%time

opt\_sim = optimize.dual\_annealing(D, bounds=[[-10, 10]] \* 4)

num\_of\_iterations, func\_value, func\_count = opt\_sim.nit, round(opt\_sim.fun, 3), opt\_sim.nfev

results.loc[algs[2], :] = [num\_of\_iterations, func\_value, func\_count]

opt\_evol = optimize.differential\_evolution(D, bounds=[[-10, 10]] \* 4, maxiter=1000)

num\_of\_iterations, func\_value, func\_count = opt\_evol.nit, round(opt\_evol.fun, 3), opt\_evol.nfev

results.loc[algs[3], :] = [num\_of\_iterations, func\_value, func\_count]

**# Results visualization**

titles = ['Nelder-Mead algorithm approximation', 'Levenberg-Marquardt algorithm approximation', 'Simulated Annealing approximation', 'Differential evolution algorithm approximation']

algorithms = [opt\_Nelder, opt\_LM, opt\_sim, opt\_evol]

fig, ax = plt.subplots(figsize=(20, 10))

plt.style.use('seaborn')

ax.scatter(X, Y, color='k')

for title, algorithm in zip(titles, algorithms):

    ax.plot(X, [approx(x, \*algorithm.x) for x in X], linewidth = 2, label = title)

ax.legend()

ax.set\_title('Approximation results')

**# Results**

results

writer = pd.ExcelWriter('results.xlsx')

results.to\_excel(writer)

writer.save()