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

Mike Chumakov

J4132c

Accepted by

Dr Petr Chunaev

St. Petersburg

2022

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 (x_k, y_k), *where* $k = 0, \dots, 1000$, *according to the rule:*

$$y_k = \begin{cases} -100 + \delta_k, & f(x_k) < -100, \\ f(x_k) + \delta_k, & -100 \le f(x_k) \le 100, \\ 100 + \delta_k, & f(x_k) > 100, \end{cases} \qquad x_k = \frac{3k}{1000},$$

$$f(x) = \frac{1}{x^2 - 3x + 2},$$

where $\delta_k \sim N(0,1)$ *are values of a random variable with standard normal distribution. Approximate the data by the rational function*

$$F(x, a, b, c, d) = \frac{ax + b}{x^2 + cx + d}$$

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

$$D(a, b, c, d) = \sum_{k=0}^{1000} (F(x_k, a, b, c, d) - y_k)^2.$$

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.).*

Brief theoretical part

Metaheuristic algorithms are natural-inspired algorithms that solve an optimization problem by trial and error. Generally speaking, metaheuristic methods do not guarantee that a solution to the optimization problem will be found.

Within the framework of this laboratory work, the method of differential evolution is considered. Differential evolution is a metaheuristic algorithm that solves the optimization problem through the evolution of the population of agents, that is, possible solutions, creating new generations of agents by combining existing and a further selection of the best ones.

It is proposed to first generate noisy data generated by an essentially rational function with discontinuities on the segment [0, 1], and then find a rational function F of the corresponding form approximating them. An extremely nonlinear problem of numerical optimization arises, where the chosen initial approximation can have a significant effect on the result.

Results

Minimization by Levenberg-Marquart method

The initial approximations a, b, c, d for the Levenberg-Marquart method are equal to a =-1, b =1, c=1, d=1. The values of the coefficients after minimization are a = 0.98 and b = 3.17, c = 1.09, d = 0.72 number of function evaluations: 166, $\varepsilon = 0.01$ *. When choosing such initial approximations, the graph of the function was approximated as in Figure 1.*

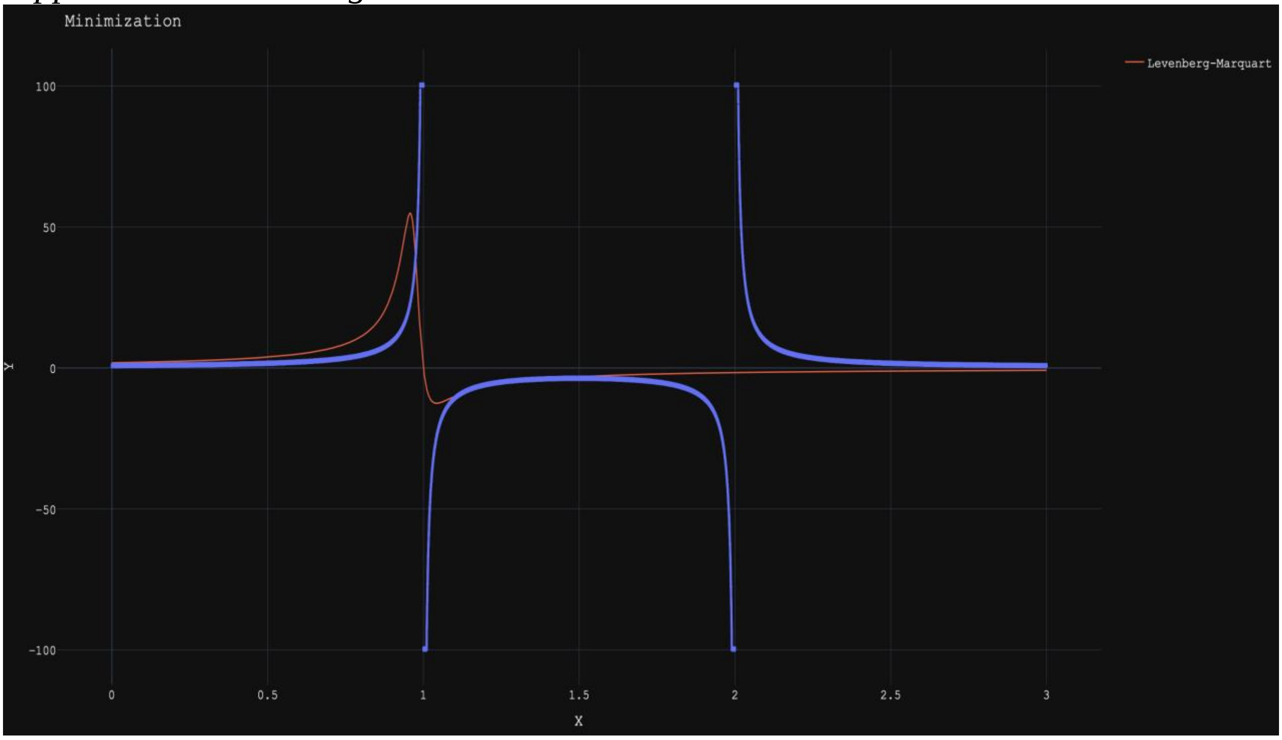

Figure 1: Rational function approximation of the least squares method with minimization by Levenberg-Marquart method

Minimization by Nelder-Mead method

The initial approximations a, b, c, d for the Nelder-Mead method method are equal to a = 0, b = 0, c = 0, d = 0. The values of the coefficients after minimization are a = -1.49 and b =1.49, c =-2.00, d = 1.00 number of function evaluations: 456, number of iteration: 259, $\varepsilon = 0.001$ *. When choosing such initial approximations, the graph of the function was approximated as in Figure 2.*

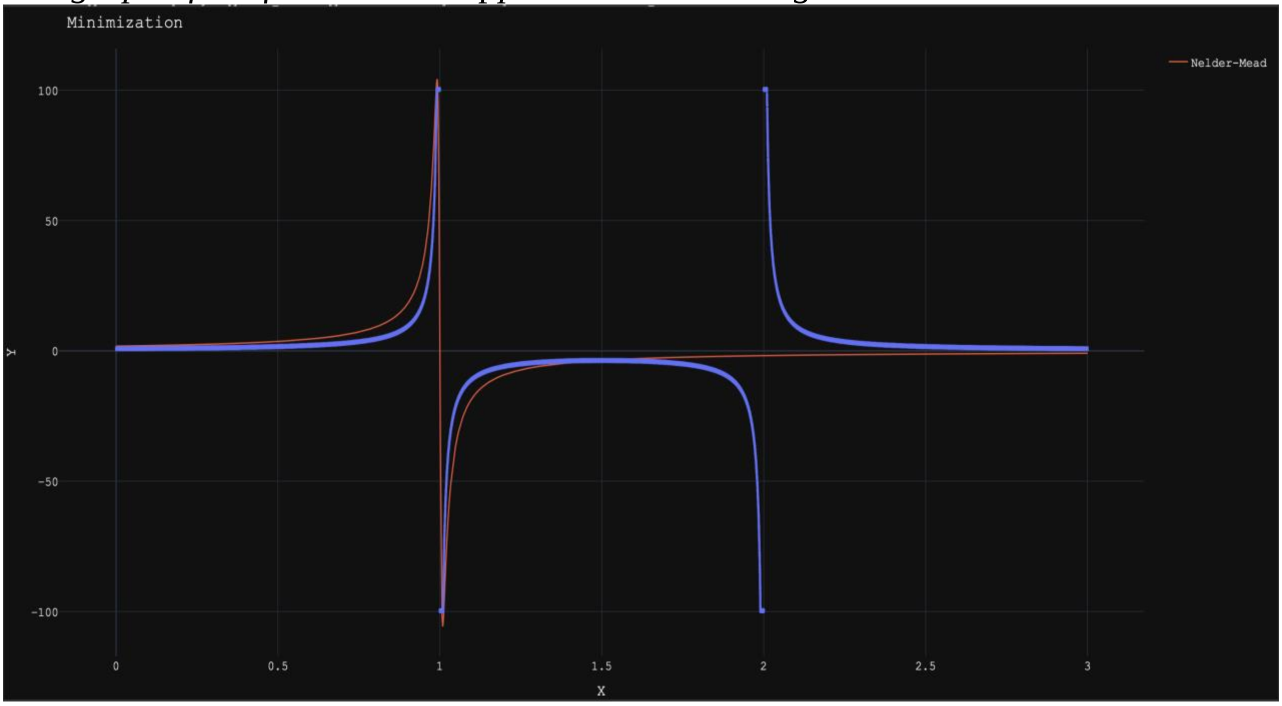

Figure 2: Rational function approximation of the least squares method with minimization by Nelder-Mead method

Minimization by Differential Evolution method

The initial approximations a, b, c, d for the Differential Evolution method method in bounds a = (-2,2), b = (-2,2), c = (-2,2), d = (-2,2). The values of the coefficients after minimization are a = -0.99 and b =1.00, c =-2.00, d = 1.00 number of function evaluations: 8825, number of max iteration: 144, $\varepsilon = 0.001$ *. When choosing such initial approximations, the graph of the function was approximated as in Figure 3.*

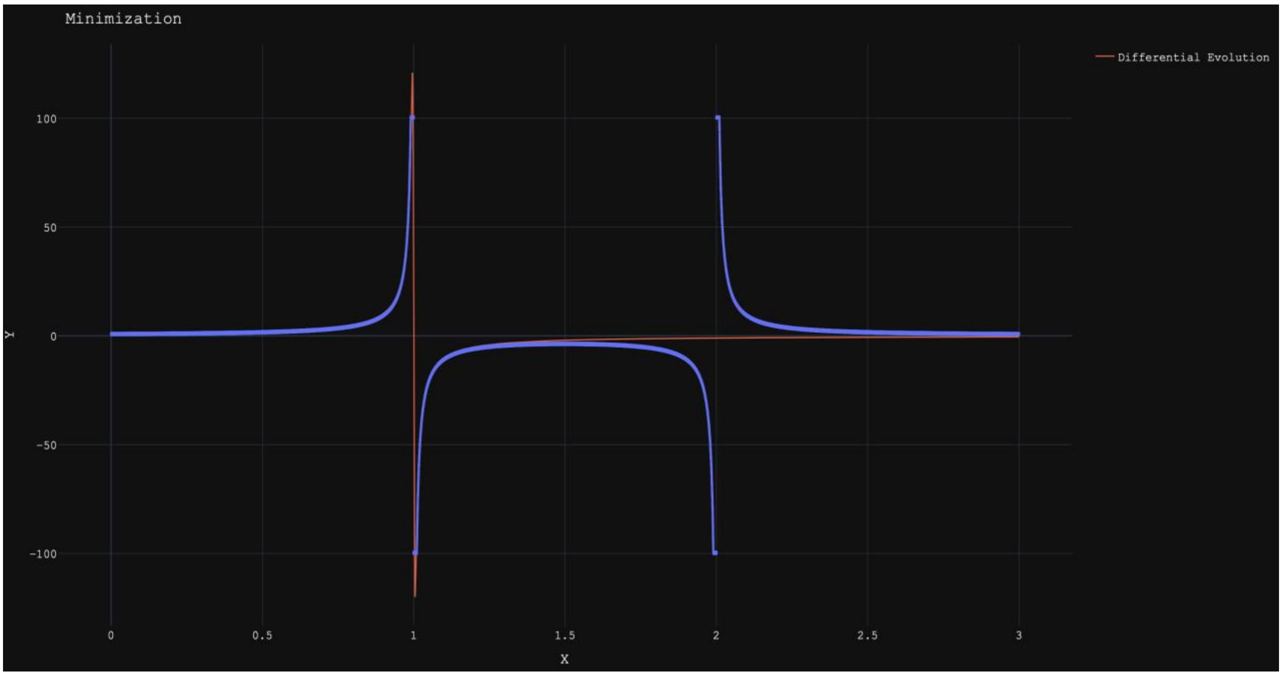

Figure 3: Rational function approximation of the least squares method with minimization by Differential Evolution method

Minimization results on the one plot

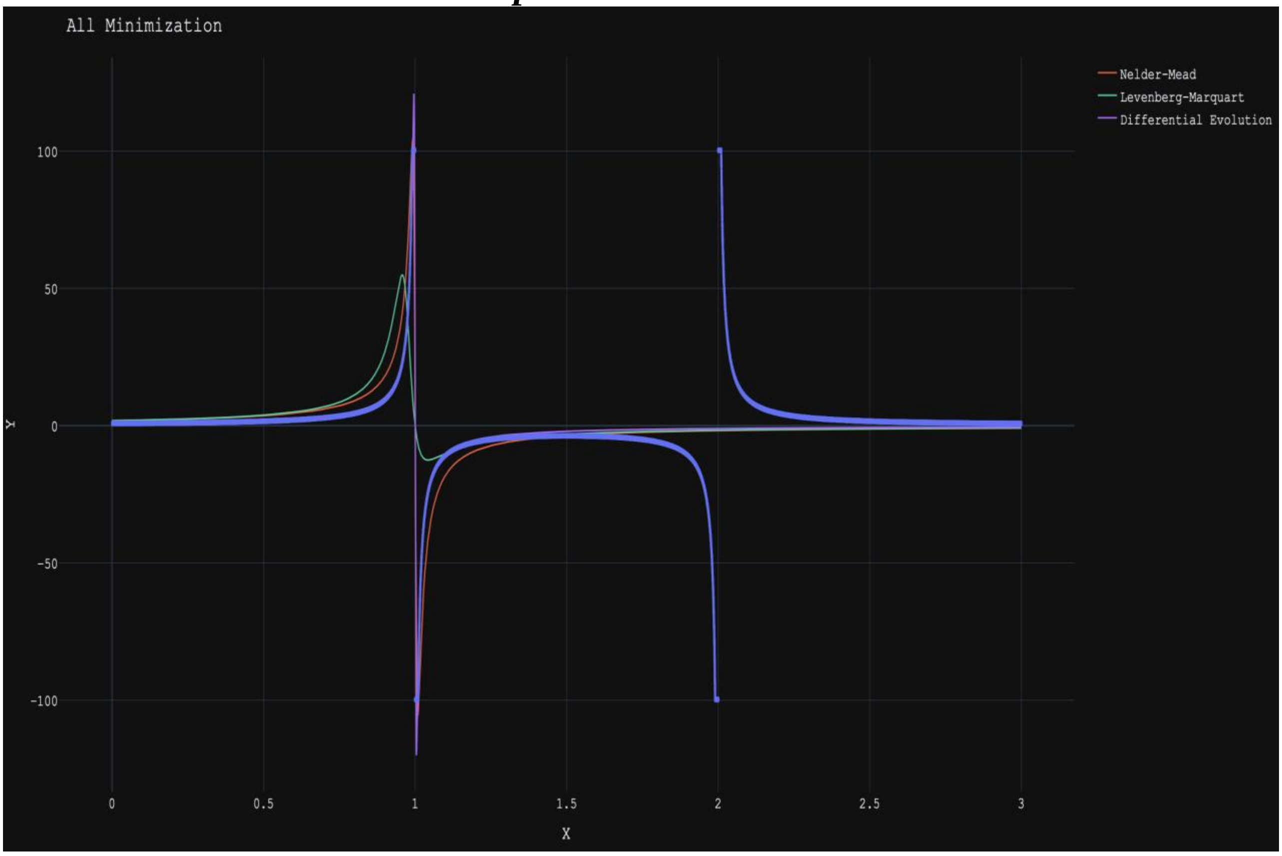

Figure 4: Rational function approximation of the least squares method with minimization by Differential Evolution, Nelder-Mead and Levenberg-Marquart methods

Conclusions

In the course of the laboratory work use of stochastic and metaheuristic algorithms Differential Evolution in the tasks of unconstrained nonlinear optimization and the experimental comparison of them with Nelder-Mead and Levenberg-Marquardt algorithms. The Levenberg-Marquart method performed worse than others, but if you remove the restrictions for epsilan, the graph will be similar to the results of other methods. The Nelder-Mead and Levenberg-Marquart methods have similar results, but the number of evaluations of the function is very different (see Figure 4). The result is also strongly influenced by restrictions and the choice of initial approximations.

Appendix

Code Link

LAB 4

```
import random
import numpy as np

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


def y(x,delta):
  if fx(x) < -100:
    return -100 + delta

  if abs(fx(x)) <= 10**2:
    return fx(x) + delta

  if fx(x) > 100:
    return 100 + delta

xk = np.zeros(1000)
yk = np.zeros(1000)
delta = np.random.sample(1000)

for k in range(0,1000):
  xk[k] = (3*k)/1000
  yk[k] = y(xk[k],delta[k])
```

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

def plot_func(x,y,title,label,fs=26,flag=False, additionalX=None, additionalY=None,addi
  fig, ax = plt.subplots()
  ax.scatter(x, y, label=label)
  # if flag:
  #   plt.plot(additionalX, additionalY, label = additionallabel)
  ax.set_title(title,fontsize = fs)
  ax.set_xlabel('X', fontsize=fs)
  ax.set_ylabel('Y', fontsize=fs)
  plt.tick_params(axis='both', which='major', labelsize=fs)
  fig.set_figwidth(15)
  fig.set_figheight(15)
  # plt.plot(x,y)
  # plt.axis([-0.1, 1.01, 0, 2.3])
  handles, labels = ax.get_legend_handles_labels()
  ax.legend(handles, labels, prop={'size': 26} )
  plt.show()
```

```
plot_func(xk,yk,"data","data")
```

```
def rsm(x,n):
  return (a * xk + b)/(xk**2+c*xk+d)
```

```
from scipy import optimize
def f1(z,*args):
  a,b,c,d = z
  return (a * xk + b)/(xk**2+c*xk+d)

def f(z,*args):
  a,b,c,d = z
  return ((f1(z)-yk)**2)

res_lm = optimize.least_squares(f,[-1,1,1,1],method='lm',xtol = 0.001)
res_lm.x
# coef = optimize.curve_fit(lsm, xk, yk, method=None)
# a = coef[0][0]
# b = coef[0][1]
# print(a)
# print(b)

```

```
res_lm.nfev
```

```
def f1(z,*args):
  a,b,c,d = z
  return (a * xk + b)/(xk**2+c*xk+d)

def f(z,*args):
  a,b,c,d = z
  return np.sum(((f1(z)-yk)**2))

x0=np.zeros(4)
res_nm = optimize.minimize(f,x0,args=(xk,yk), method='Nelder-Mead', tol=0.001,  options
res_nm.x
```

```
res_nm.nit
```

```
from scipy.optimize import rosen, differential_evolution
bounds = [(-2,2), (-2, 2), (-2, 2), (-2, 2)]
result = differential_evolution(f, bounds, maxiter=1000, tol=0.001)
result.x

```

```
result.nfev
```

```
  fig, ax = plt.subplots()
```

```
plt.plot(xk, rsm(xk,res_lm.x[0],res_lm.x[1],res_lm.x[2],res_lm.x[3]), label =  Levenb
plt.plot(xk, rsm(xk,res_nm.x[0],res_nm.x[1],res_nm.x[2],res_nm.x[3]), label = "Nelder
plt.plot(xk, rsm(xk,result.x[0],result.x[1],result.x[2],result.x[3]), label = 'Differ
ax.scatter(xk, yk)
# if flag:
#   plt.plot(additionalX, additionalY, label = additionallabel)
# ax.set_title(title,fontsize = fs)
ax.set_xlabel('X', fontsize=26)
ax.set_ylabel('Y', fontsize=26)
plt.tick_params(axis='both', which='major', labelsize=26)
fig.set_figwidth(15)
fig.set_figheight(15)
# plt.plot(x,y)
# plt.axis([-0.1, 1.01, 0, 2.3])
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels, prop={'size': 26} )
plt.show()
```