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

Abizer Safdari

J4134c

Accepted by

Dr Petr Chunaev

St. Petersburg

2020

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

- Stochastic optimization refers to a collection of methods for minimizing or maximizing an objective function when randomness is present. Over the last few decades these methods have become essential tools for science, engineering, business, computer science, and statistics.
- Randomness usually enters the problem in two ways: through the cost function or the constraint set. Although stochastic optimization refers to any optimization method that employs randomness within some communities, we only consider settings where the objective function or constraints are random.
- A metaheuristic is a higher-level procedure or heuristic designed to find, generate, or select a heuristic that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity. Metaheuristics may make relatively few assumptions about the optimization problem being solved and so may be usable for a variety of problems. Compared to optimization algorithms and iterative methods, metaheuristics do not guarantee that a globally optimal solution can be found on some class of problems.[3] Many metaheuristics implement some form of stochastic optimization, so that the solution found is dependent on the set of random variables generated.[2] In combinatorial optimization, by searching over a large set of feasible solutions, metaheuristics can often find good solutions with less computational effort than optimization algorithms, iterative methods, or simple heuristics.
- The Nelder-Mead method uses a geometrical shape called a simplex as its ‘vehicle’ of sorts to search the domain. This is why the technique is also called the Simplex search method. In layman’s terms, a simplex is the n-dimensional version of a ‘triangle’. Nelder-Mead starts off with a randomly-generated simplex (We will discuss how, later). At every iteration, it proceeds to reshape/move this simplex, one vertex at a time, towards an optimal region in the search space. During each step, it basically tries out one or a few modifications to the current simplex, and chooses one that shifts it towards a ‘better’ region of the domain. In an ideal case, the last few iterations of this algorithm would involve the simplex shrinking inwards towards the best point inside it. At the end, the vertex of the simplex that yields that most optimal objective value, is returned.
- Levenberg–Marquardt algorithm, also known as the damped least-squares method, is used to solve non-linear least squares problems. These minimization problems arise especially in least squares curve fitting. It finds only a local minimum, which is not necessarily the global minimum. The method interpolates between the Gauss–Newton algorithm and the method of gradient descent. The method is more robust than the Gauss Newton,

which means that in many cases it finds a solution even if it starts very far off the final minimum

- **Simulated annealing** (**SA**) is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space for an optimization problem. It is often used when the search space is discrete. For problems where finding an approximate global optimum is more important than finding a precise local optimum in a fixed amount of time, simulated annealing may be preferable to exact algorithms such as gradient descent, Branch and Bound.
- Simulated annealing can be used for very hard computational optimization problems where exact algorithms fail; even though it usually achieves an approximate solution to the global minimum, it could be enough for many practical problems.
- **differential evolution** (**DE**) is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Such methods are commonly known as metaheuristics as they make few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, metaheuristics such as DE do not guarantee an optimal solution is ever found.
- DE is used for multidimensional real-valued functions but does not use the gradient of the problem being optimized, which means DE does not require the optimization problem to be differentiable. DE can therefore also be used on optimization problems that are not even continuous, are noisy, change over time, etc

Results

Method	Function	Least square error	Iterations	F-calculations
Nemled-mend	$\frac{ax+b}{x^2+cx+d}$	215063.617626	202	356
Levenberg Marquardt	$\frac{ax+b}{x^2+cx+d}$	133455.102165	4	12
Simulated Annealing	$\frac{ax+b}{x^2+cx+d}$	137149.858814	4	963
Differential Evolution	$\frac{ax+b}{x^2+cx+d}$	140082.911800	4	1540

Table 1: Number of iteration, least square error and rational approximation functions using stochastic and metaheuristic algorithms.

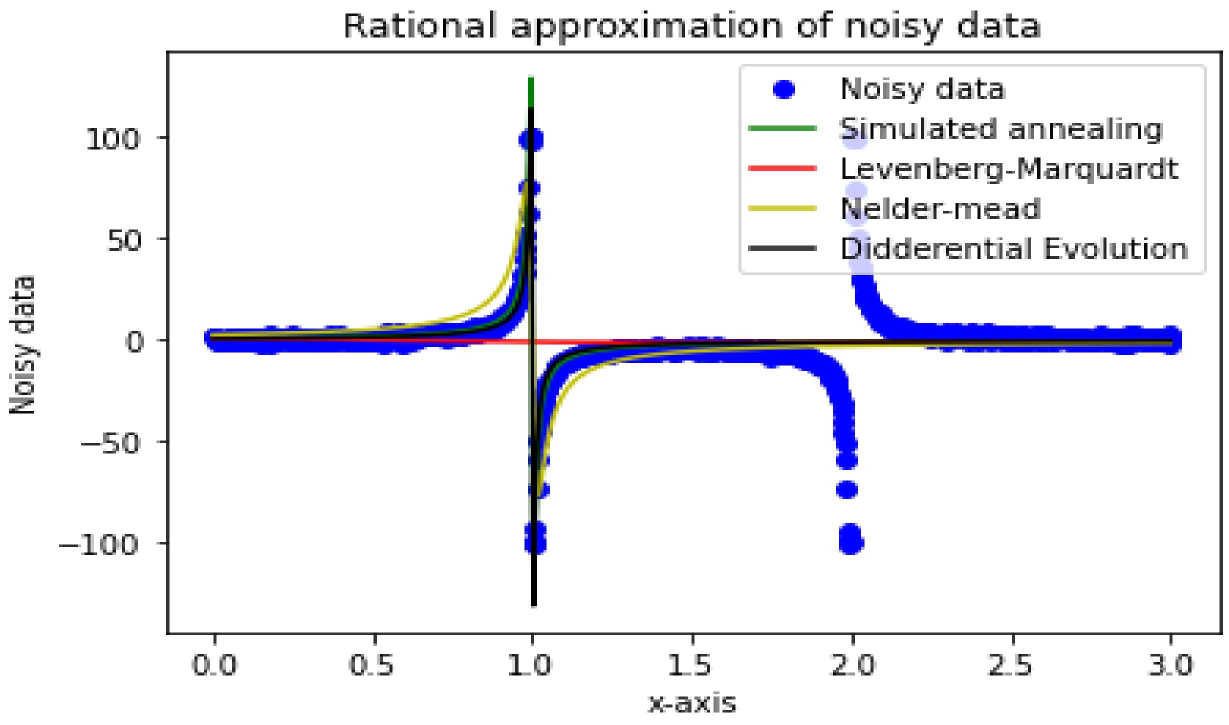

Figure 1: The rational approximation of the noisy data using Nelder-mead, Levenberg-Marquardt algorithm, Simulated annealing, Differential evolution.

Conclusions

- Levenberg-Marquardt algorithm is only applicable for functions which are continuous and differentiable. But as the noisy data has uncertainties due to which continuous function may not be appropriate approximation for the noisy data. To get the best approximation function stochastic and metaheuristic algorithms are preferable.
- As there are uncertainties in the data, deterministic algorithms like Nelder-mead and Levenberg-Marquardt cannot work properly, where as in Stochastic and metaheuristic algorithms, uncertainty is considered under assumptions of assumed probability distributions. So, they give accurate approximation for the global optimization. But the probability of finding global optimization is not always 1.
- Stochastic algorithms optimize the function in less iterations and there are no complex computations compared to Levenberg-Marquardt algorithm. But the functional evaluations are higher for the required precision.
- Differential evolution and Simulated annealing give similar results, but Simulated annealing works significantly slower. Nelder-mead and Levenberg Marquardt algorithm have found some local minimum, but not very optimal. Thus, Differential evolution has the best score from these algorithms.

Appendix

```
"""
Created on Sat Oct 17 20:35:50 2020

@author: abizer
"""
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt

# generating noise data
xk=np.arange(0, 3.003, 0.003)
fk=1/(xk**2-3*xk+2)
yk=np.zeros(1001)
for i in range(1001):
    if fk[i]< -100:
        yk[i]=-100+np.random.normal(0,1)
    elif fk[i]> 100:
        yk[i]=100+np.random.normal(0,1)
    else:
        yk[i]=fk[i]+np.random.normal(0,1)

# defining rational approximatin function.
def fun(x):
    fun=(x[0]*xk+x[1])/(xk**2+x[2]*xk+x[3])
    return fun

# defining least squares of rational approximation function
def D(x):
    return sum((fun(x)-yk)**2)

# defining residual function
def residual(x):
    return fun(x)-yk

#defining jacobian matrix for rational approximation function
def jacobian(x):
    j=np.empty((xk.size, x.size))
    den=(xk**2+x[2]*xk+x[3])
    j[:,0]=xk/den
    j[:,1]=1/den
    j[:,2]=-xk*fun(x)/den
    j[:,3]=-fun(x)/den
    return j
```

```
# intial approximations
x0=np.array([0.5,0.5,1,1])

# caluclating Rational approximation using Nelder mead.
res_nm=optimize.minimize(D, x0, method='Nelder-Mead',\
                    options={'maxiter':1000, 'disp':True, 'fatol':0.001} )

   #caluclating rational approximation using evenberg-Marquardt algorithm
res_lm=optimize.least_squares(residual, x0, jac=jacobian,\
                        method='lm',ftol=0.001 )

# caluclating rational apprximation using simulated anneling.
lw = [-10] * 4
up = [10] * 4
bounds=list(zip(lw, up))
res_sm = optimize.dual_annealing(D, bounds,maxiter=4, accept=1, \
 seed=1234 )

# caluclating rational approximation using differential evalution.
res_de=optimize.differential_evolution(D, bounds, maxiter=1000, atol=0.001)

# Caluclating rational approximation using particle swarn optimizaion
bounds = [(-10, 10), (-10, 10), (-10, 10), (-10, 10)] # upper and lower bounds of
variables
nv = 4 # number of variables

y_nm=fun(res_nm.x)
y_lm=fun(res_lm.x)
y_sm=fun(res_sm.x)
y_de=fun(res_de.x)

# plotting approximation function
plt.plot(xk, yk,'bo' ,label='Noisy data')
plt.plot(xk, y_sm, '-g', label='Simulated annealing')
plt.plot(xk, y_lm, '-r', label='Levenberg-Marquardt')
plt.plot(xk, y_nm, '-y' , label='Nelder-mead')
plt.plot(xk, y_de, '-k', label='Didderential Evolution')
plt.title('Rational approximation of noisy data')
plt.ylabel('Noisy data')
plt.xlabel('x-axis')
plt.legend()
plt.show()
```