computer sciences and mathematics forum

Proceeding Paper

Non-linear optimization method for maximum point search in functions with corner or cusp points †

Pasquale Fotia *[1], **Massimiliano Ferrara** [2]

Department of Law, Economics and Human Sciences , University Mediterranea of Reggio Calabria; pasquale.fotia.digies@unirc.it

[2] Department of Law, Economics and Human Sciences & Decisions Lab, University Mediterranea of Reggio Calabria, ICRIOS – Department of Management and Technology, Bocconi University, Milan; massimiliano.ferrara@unirc.it; massimiliano.ferrara@unibocconi.it

* Correspondence: pasquale.fotia.digies@unirc.it

† Presented at the IOCMA2023 - S4. Financial Mathematics.

Abstract: A function is non-differentiable when there is a cusp or a corner point in its graph. To solve this problem we propose a nonlinear optimization model whose objective function is the Euclidean distance function. To identify the maximum points of a function that has corner or cusp points, according to the proposed model, a series of segments are generated which are measured through the Euclidean distance, which are all perpendicular to the abscissa axis. Therefore, by maximizing the Euclidean distance it is possible to identify the segment whose points represent the maximum of the function and its projection on the abscissa axis. The proposed model therefore wants to be an alternative to maximum point search methods in the presence of functions that have points of non-derivability.

Keywords: non-linear problem; optimization; non-derivability

Citation: Tiwari, S. Prediction Models Based on Clustering. *Comput. Sci. Math. Forum* **2023**, 2, x. https://doi.org/10.3390/xxxxx

Academic Editor(s):

Published: 28 April 2023

1. Introduction

This study presents a novel method for optimizing functions with angular points or cusps. Several disciplines, such as physics, engineering, finance, and machine learning, encounter this type of issue frequently. The method given reformulates the optimization problem with Euclidean distance as the objective function. The approach is tested with two empirical examples: one employing the absolute value function with a single independent variable and the other with two independent variables. Python's SciPy module is utilized to implement the stated optimization problems. The outcomes are assessed using two parameters supplied by the optimization solver, specifically the SLQLP algorithm that minimizes the objective function. While the suggested method does not considerably outperform existing techniques, it does offer a viable option for optimizing functions with sharp corners or cusps. Further research could investigate the exact circumstances in which the proposed method performs better, or if there are mathematical, non-heuristic alternatives that could be more effective with this reformulation. Such insights could enhance the efficacy of the proposed strategy and make it a more competitive alternative.

2.Methods

If we consider a function with positive variables and we take into account the segments that go from the x-axis to the function values, which are internal to the area below the function and above the x-axis, then if we identify the longest segment, one of its endpoints will be equal to the maximum value of the function. From this idea, we can maximize the Euclidean distance of these segments.

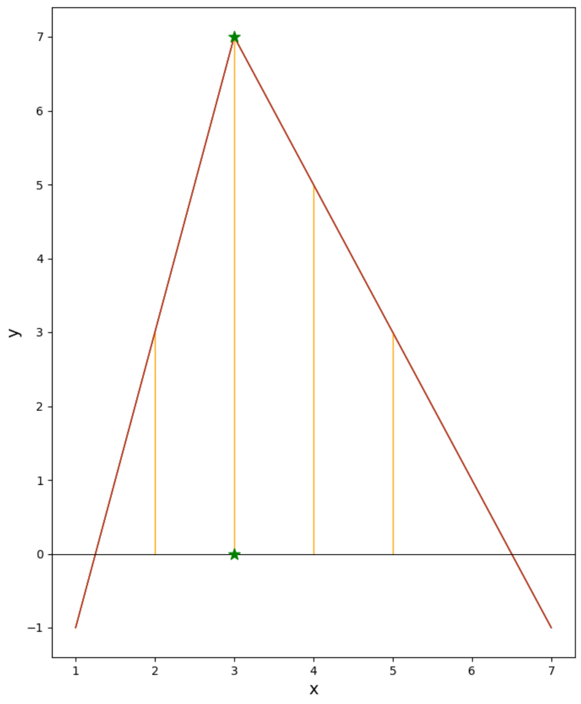


Figure 1. Graphical representation of the function y = - 3|x-3| + x + 4, plotted over the range of x from 1 to 7. The vertical segments represent the distances from the x-axis to the function, between x=2 and x=5. A green asterisk denotes the maximum point of the function at (x=3, y=7), while the horizontal black line represents the x-axis.

In this study, we empirically evaluate the effectiveness of a reformulation technique for an optimization problem using the functions 1 and 2.

$$y = -3|3 - x| + 4 + x \tag{1}$$

$$z = -\frac{|2 - x| + |2 - y|}{6} + 4 \tag{2}$$

Below are shown the two optimization problems, 3 and 4, in which the two functions, 1 and 2, are present, which become constraints of the problems, and to respect the orthogonality of the segments we add the equality constraints between the variables that identify the extremum of the segment that is not part of the function of which we are finding the maximum point.

$$\begin{aligned} \min \quad & -\sqrt{(x_1 - x_0)^2 + x_2^2} \\ s.t. \quad & -3|3 - x_0| - x_2 + 4 + x_0 = 0 \\ & x_2 \geq 0 \\ & x_0 - x_1 = 0 \end{aligned} \tag{3}$$

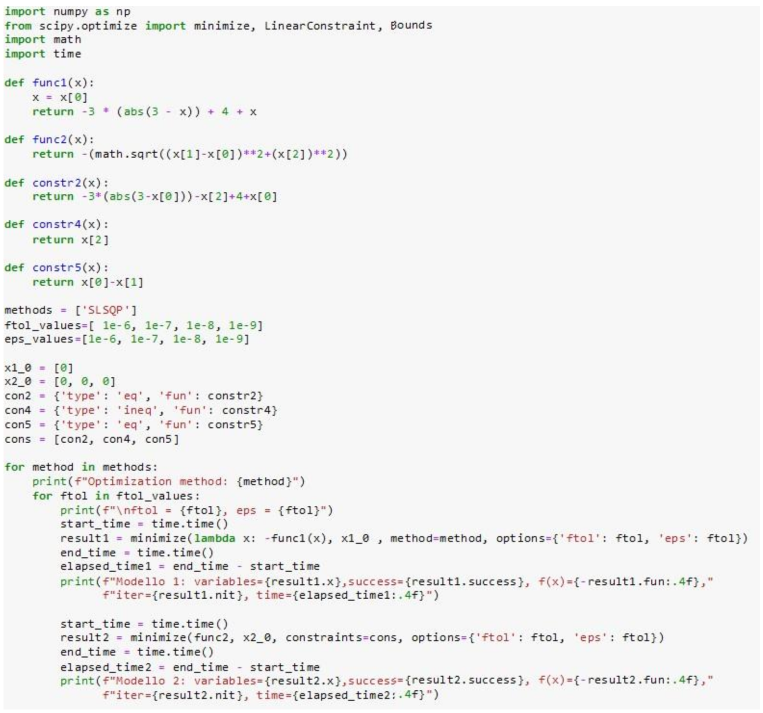
$$\begin{aligned} min \quad & -\sqrt{(x_1 - x_0)^2 + (x_3 - x_2)^2 + x_4^2} \\ s.t. \quad & -\frac{|2 - x_0| + |2 - 2x_2|}{6} + 4 - x_4 \\ & x_4 \geq 0 \\ & x_0 - x_1 = 0 \\ & x_2 - x_3 = 0 \end{aligned} \tag{4}$$

2.1. Python implementation

This section shows how the whole process has been implemented in python using the SciPy library. SciPy is a Python collection of numerical routines that provides the core building blocks for modelling and solving scientific problems. SciPy supports methods for optimization, integration, interpolation, eigenvalue problems, algebraic equations, and differential equations, as well as specific data structures such as sparse matrices and k-dimensional trees [2].

Optimization problems 3 and 4 are developed in python as in figure 2 and 3 respectively. The code in Figure 2, defines two functions, func1 and func2, and three constraint functions, constr2, constr4, and constr5. Func1 takes a one-dimensional input x and returns the result of an expression involving x. Func2 takes a three-dimensional input x and returns the result of a mathematical expression involving the components of x. constr2, constr4, and constr5 are constraint functions that take a three-dimensional input x and return the result of an expression that constrains the optimization problem.

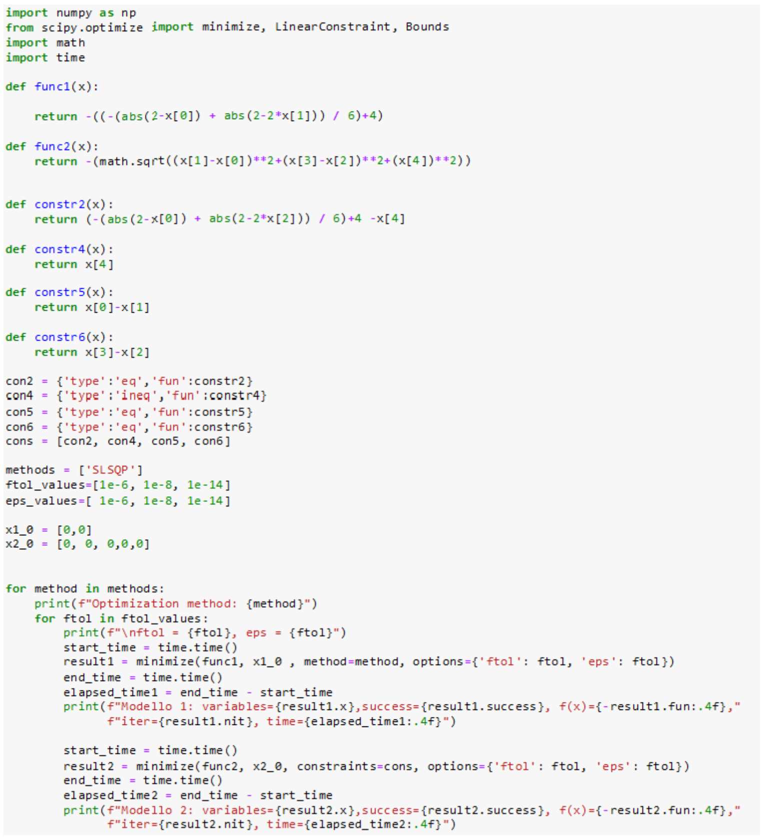
The optimization is performed using the minimize function from the scipy.optimize module, with the SLSQP method [1]. Two sets of optimizations are performed: the first with func1 as the objective function (1), and the second with func2 and the three constraint functions as constraints. The optimization is run multiple times with different values of ftol and eps, and the results are printed to the console.



```python
import numpy as np
from scipy.optimize import minimize, LinearConstraint, Bounds
import math
import time

def func1(x):
    x = x[0]
    return -3 * (abs(3 - x)) + 4 + x

def func2(x):
    return -(math.sqrt((x[1]-x[0])**2+(x[2])**2))

def constr2(x):
    return -3*(abs(3-x[0]))-x[2]+4+x[0]

def constr4(x):
    return x[2]

def constr5(x):
    return x[0]-x[1]

methods = ['SLSQP']
ftol_values=[ 1e-6, 1e-7, 1e-8, 1e-9]
eps_values=[1e-6, 1e-7, 1e-8, 1e-9]

x1_0 = [0]
x2_0 = [0, 0, 0]
con2 = {'type': 'eq', 'fun': constr2}
con4 = {'type': 'ineq', 'fun': constr4}
con5 = {'type': 'eq', 'fun': constr5}
cons = [con2, con4, con5]

for method in methods:
    print(f"Optimization method: {method}")
    for ftol in ftol_values:
        print(f"\nftol = {ftol}, eps = {ftol}")
        start_time = time.time()
        result1 = minimize(lambda x: -func1(x), x1_0 , method=method, options={'ftol': ftol, 'eps': ftol})
        end_time = time.time()
        elapsed_time1 = end_time - start_time
        print(f"Modello 1: variables={result1.x},success={result1.success}, f(x)={-result1.fun:.4f},"
              f"iter={result1.nit}, time={elapsed_time1:.4f}")

        start_time = time.time()
        result2 = minimize(func2, x2_0, constraints=cons, options={'ftol': ftol, 'eps': ftol})
        end_time = time.time()
        elapsed_time2 = end_time - start_time
        print(f"Modello 2: variables={result2.x},success={result2.success}, f(x)={-result2.fun:.4f},"
              f"iter={result2.nit}, time={elapsed_time2:.4f}")
```

Figure 2. Python implementation of optimization problem 3 using SciPy library.

In Figure 3, the code performs the same functions as the first in Figure 2 but more constraints are inserted and the problem is developed with more variables.

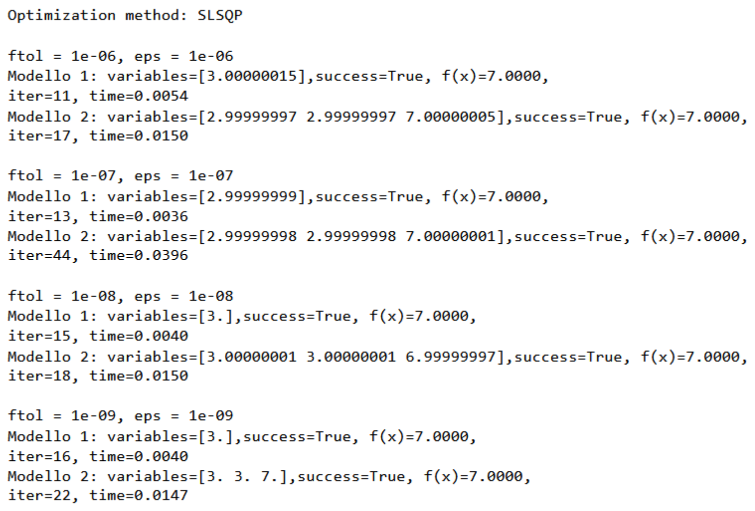


```python
import numpy as np
from scipy.optimize import minimize, LinearConstraint, Bounds
import math
import time

def func1(x):

    return -((-(abs(2-x[0]) + abs(2-2*x[1])) / 6)+4)

def func2(x):
    return -(math.sqrt((x[1]-x[0])**2+(x[3]-x[2])**2+(x[4])**2))


def constr2(x):
    return (-(abs(2-x[0]) + abs(2-2*x[2])) / 6)+4 -x[4]

def constr4(x):
    return x[4]

def constr5(x):
    return x[0]-x[1]

def constr6(x):
    return x[3]-x[2]

con2 = {'type':'eq','fun':constr2}
con4 = {'type':'ineq','fun':constr4}
con5 = {'type':'eq','fun':constr5}
con6 = {'type':'eq','fun':constr6}
cons = [con2, con4, con5, con6]

methods = ['SLSQP']
ftol_values=[1e-6, 1e-8, 1e-14]
eps_values=[ 1e-6, 1e-8, 1e-14]

x1_0 = [0,0]
x2_0 = [0, 0, 0,0,0]


for method in methods:
    print(f"Optimization method: {method}")
    for ftol in ftol_values:
        print(f"\nftol = {ftol}, eps = {ftol}")
        start_time = time.time()
        result1 = minimize(func1, x1_0 , method=method, options={'ftol': ftol, 'eps': ftol})
        end_time = time.time()
        elapsed_time1 = end_time - start_time
        print(f"Modello 1: variables={result1.x},success={result1.success}, f(x)={-result1.fun:.4f},"
              f"iter={result1.nit}, time={elapsed_time1:.4f}")

        start_time = time.time()
        result2 = minimize(func2, x2_0, constraints=cons, options={'ftol': ftol, 'eps': ftol})
        end_time = time.time()
        elapsed_time2 = end_time - start_time
        print(f"Modello 2: variables={result2.x},success={result2.success}, f(x)={-result2.fun:.4f},"
              f"iter={result2.nit}, time={elapsed_time2:.4f}")
```

Figure 3. Python implementation of optimization problem 4 using SciPy library

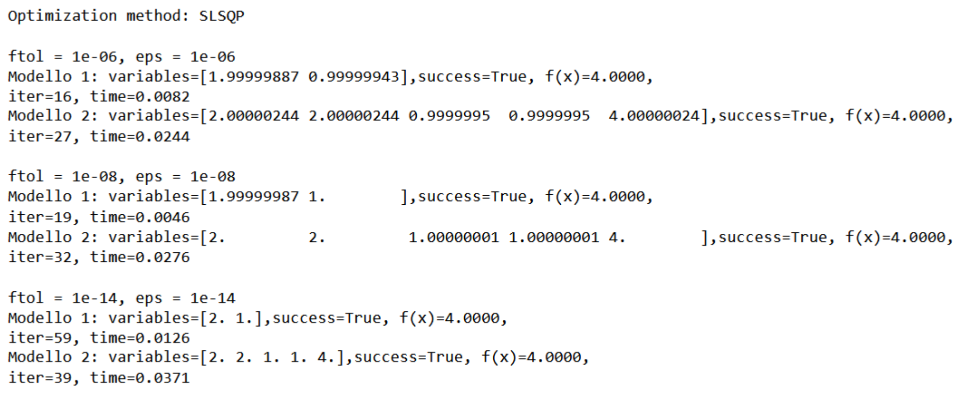
3. Results and Discussion

When the value of the 'success' parameter is true, this indicates that the optimization is successful and the proposed method has found a viable solution(figure 4 and 5). This validation method is provided by the SciPy library itself. In figures 4 and 5, the results of the two optimization problems are shown. It is observed that the number of iterations and the elapsed time are lower when the optimization is performed directly without using the proposed method in this work. However, the differences between the two methods are not significant. Moreover, it is noteworthy that for the resolution of the problem, the model formulated with the Euclidean objective function is better for the value of ftol and eps equal to 1e-14. This suggests that the choice of objective function and optimization method may depend on the specific problem and the desired level of accuracy. Therefore, it is important to carefully select the appropriate optimization method and tune the parameters to achieve the best results.



```
Optimization method: SLSQP

ftol = 1e-06, eps = 1e-06
Modello 1: variables=[3.00000015],success=True, f(x)=7.0000,
iter=11, time=0.0054
Modello 2: variables=[2.99999997 2.99999997 7.00000005],success=True, f(x)=7.0000,
iter=17, time=0.0150

ftol = 1e-07, eps = 1e-07
Modello 1: variables=[2.99999999],success=True, f(x)=7.0000,
iter=13, time=0.0036
Modello 2: variables=[2.99999998 2.99999998 7.00000001],success=True, f(x)=7.0000,
iter=44, time=0.0396

ftol = 1e-08, eps = 1e-08
Modello 1: variables=[3.],success=True, f(x)=7.0000,
iter=15, time=0.0040
Modello 2: variables=[3.00000001 3.00000001 6.99999997],success=True, f(x)=7.0000,
iter=18, time=0.0150

ftol = 1e-09, eps = 1e-09
Modello 1: variables=[3.],success=True, f(x)=7.0000,
iter=16, time=0.0040
Modello 2: variables=[3. 3. 7.],success=True, f(x)=7.0000,
iter=22, time=0.0147
```

Figure 4. Report solution problem 3 implemented in Python



```
Optimization method: SLSQP

ftol = 1e-06, eps = 1e-06
Modello 1: variables=[1.99999887 0.99999943],success=True, f(x)=4.0000,
iter=16, time=0.0082
Modello 2: variables=[2.00000244 2.00000244 0.9999995  0.9999995  4.00000024],success=True, f(x)=4.0000,
iter=27, time=0.0244

ftol = 1e-08, eps = 1e-08
Modello 1: variables=[1.99999987 1.        ],success=True, f(x)=4.0000,
iter=19, time=0.0046
Modello 2: variables=[2.         2.         1.00000001 1.00000001 4.        ],success=True, f(x)=4.0000,
iter=32, time=0.0276

ftol = 1e-14, eps = 1e-14
Modello 1: variables=[2. 1.],success=True, f(x)=4.0000,
iter=59, time=0.0126
Modello 2: variables=[2. 2. 1. 1. 4.],success=True, f(x)=4.0000,
iter=39, time=0.0371
```

Figure 5. Report solution problem 4 implemented in Python

Some limitation of the model proposed:

1. The model assumes that the objective function has a well-defined maximum point, which may not always be the case. Some functions may have multiple local maxima or no maximum at all, making it difficult to identify the global maximum using the proposed approach.

2. The model requires the generation of a series of segments perpendicular to the abscissa axis, which can be computationally expensive for high-dimensional functions or functions with

complex shapes. This could limit the scalability of the approach for larger and more complex optimization problems.

4. Conclusion

In conclusion, further tests and developments are needed to understand if this formulation actually favors a better search for the optimal solution. It is important to explore different combinations of values for ftol and eps and to test the model on a broader range of optimization problems. By doing so, it will be possible to better evaluate the effectiveness and efficiency of this formulation and its potential for future applications.

References

1. Kraft, D. (1988). A software package for sequential quadratic programming. Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt.
2. Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... & Van Mulbregt, P. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature methods, 17(3), 261-272.