UTILIZING PYTHON FOR FINDING LOCAL EXTREMA OF MULTIVARIABLE FUNCTIONS[*]

VALENTIN GEORGIEV, ATANAS ILCHEV, BOYAN ZLATANOV

ABSTRACT: *The application of the Python programming language for finding local extrema of functions of two variables highlights the advantages of modern information technologies in performing complex and time-consuming algebraic computations. This article emphasizes Python's capacity to simplify and accelerate processes that would otherwise require significant manual effort through traditional mathematical methods. Two specific examples are discussed, where classical "by hand" calculations would be laborious and prone to error. The article presents a clear procedure and accompanying Python code that automates the determination of local extrema, demonstrating the efficiency and precision achievable through computational approaches. This showcases the practical benefits of integrating Python into advanced mathematical problem-solving.*

KEYWORDS: *Python, Local Extrema, Multivariable functions.*

DOI: https://doi.org/10.46687/XJXO7873

1 Introduction

The concept of mathematical competence has evolved to include the ability to work with modern software systems. By utilizing programming languages like Python, students develop crucial skills in algorithmic thinking and problem-solving, broadening their digital competence beyond the traditionally passive use of technology. Python's versatility in computational mathematics allows learners to deepen their mathematical understanding and enhance their conceptual knowledge.

The rapid advancement of information technology has introduced a wide range of tools that can tackle both simple and complex mathematical challenges. Students today have access to powerful software applications that facilitate their learning and problem-solving processes. A notable example is the widespread use of Python in the field of mathematical analysis, where it can be leveraged to automate tedious and error-prone calculations.

In recent years, the approach to teaching has undergone significant changes, as educators must now incorporate software tools into their curriculum to align with the evolving technological landscape. The future belongs to those who embrace these advancements and integrate them into education, not as mere shortcuts, but as tools that enhance the acquisition of deeper knowledge.

In this article, we will illustrate the application of Python in finding local extrema of functions with multiple variables. The process of solving such problems manually is not only labor-intensive but also carries a high risk of errors. Therefore, the article presents a Python-based procedure that automates the entire process, ensuring accuracy and efficiency. By introducing such technologies into calculus classes, students gain a significant advantage, as they are able to focus on understanding core mathematical concepts while benefiting from the precision and speed of modern computational tools.

2 Theoretical formulation and Python Code

Although finding extrema of multi-variable functions is a common problem students are taught early on in their bachelor's degree, we would like to recall the following definitions and

[*]Partially supported by Scientific Research Grant DDVU 02/91 of MON and Scientific Research Grant RD-08-75/2016 of Shumen University.

theorem.

Definition 1. [1] Suppose $f : U \to \mathbb{R}$ where $U \subseteq \mathbb{R}^p, p \in \mathbb{N}$ and the partial derivatives of f all exist. Define the gradient of f denoted by $\nabla f(\mathbf{x})$ to be the vector

$$\nabla f(\mathbf{x}) = \left(f_{x_1}(\mathbf{x}), f_{x_2}(\mathbf{x}), \ldots, f_{x_p}(\mathbf{x})\right).$$

Definition 2. [1] The matrix $\mathbf{H}(\mathbf{x})$ whose ij^{th} entry at the point $\mathbf{x}$ is $\frac{\partial^2 f}{\partial x_i \partial x_j}$ is called the Hessian matrix.

Definition 3. [1] Suppose $f : U \to \mathbb{R}$ where $U \subseteq \mathbb{R}^p, p \in \mathbb{N}$. A point $\mathbf{x} \in U$ is called a local minimum if $f(\mathbf{x}) \leq f(\mathbf{y})$ for all $\mathbf{y} \in U$ sufficiently close to $\mathbf{x}$. A point $\mathbf{x} \in U$ is called a local maximum if $f(\mathbf{x}) \geq f(\mathbf{y})$ for all $\mathbf{y} \in D(f)$ sufficiently close to $\mathbf{x}$. A local extremum is a point of U which is either a local minumum or a local maximum. If there exists a direction in which when f is evaluated on a line through $\mathbf{x}$ having this direction and the resulting function of one variable as a local minimum and there exists a different direction in which when f is evaluated on the line through $\mathbf{x}$ with this direction, the resulting function of one variable has a local maximum, $\mathbf{x}$ is called a saddle point of f.

Theorem 1. [1] Let $f : U \to \mathbb{R}$ for U an open set in $\mathbb{R}^p, p \in \mathbb{N}$, f be a C^2 function and suppose that at some $\mathbf{x} \in U, \nabla f(\mathbf{x}) = \mathbf{0}$. Alse let μ and λ be respectively the largest and smallest eigenvalues of the matrix $\mathbf{H}(\mathbf{x})$. If $\lambda > 0$, then f has a local minimum at $\mathbf{x}$. If $\mu < 0$, then f has a local maximum at $\mathbf{x}$. If $\lambda < 0$ and $\mu > 0$, then $\mathbf{x}$ is a saddle point for f. If either λ or μ equals zero, the test fails.

Usually Theorem 1 is not used on its own, but instead the definiteness of the Hessian is analysed: positive definiteness leads to a local minimum, negative definiteness – to a maximum, neither leads to a saddle, and if the determinant of the Hessian being zero, the test is inconclusive. In practice, often the following theorem is used:

Theorem 2. [2] The symmetric matrix $\mathbf{A}$ is

- positive definite if and only if all its leading principal minors are positive;
- negative definite if and only if its leading principal minors alternate their signs, beginning with a negative sign.

Seeing as $\mathbf{H}(\mathbf{x})$ is symmetric, this criterion usually leads to simpler computations. However, doing these computations by hand is still error-prone and could lead to losing track of the main idea behind the process. In order to avoid that, one could use the following Python code:

```python
import sympy as sym

def local_extrema(f,x):
    print("f =")
    sym.pprint(f)
    f_grad = tuple(sym.diff(f,y) for y in x)
    print("Gradient of f =")
    sym.pprint(f_grad)
    critical_points = sym.solve(f_grad)
    if type(critical_points)==dict:
```

```
11          critical_points = [critical_points]
12      print("Critical Points : ",end = ’’)
13      for point in critical_points:
14          print(point, end = ", ")
15      hessian = sym.hessian(f,x)
16      print("\nHessian of f =")
17      sym.pprint(hessian)
18      for crit_p in critical_points:
19          print(f"\nTest for {crit_p}.")
20          eigenvalues = list(map(lambda x: x.evalf(chop = True), list(
    hessian.subs(crit_p).eigenvals().keys())))
21          m,l = max(eigenvalues), min(eigenvalues)
22          print(f"m = {m}, l = {l}")
23          if l > 0:
24              print(f"The smallest eigenvalue l = {l} > 0, therefore we
    have a local minimum f_min = {f.subs(crit_p)}.")
25          elif m < 0:
26              print(f"The biggest eigenvalue m = {m} < 0, therefore we
    have a local maximum f_max = {f.subs(crit_p)}.")
27          elif l<0 and m > 0:
28              print(f"The smallest eigenvalue l = {l} < 0, whereas the
    biggest one m = {m} > 0, therefore we have a saddle point f_saddle =
     {f.subs(crit_p)}.")
29          else:
30              print(f"The {’smallest’ if l==0 else ’biggest’} eigenvalue
    is equal to zero. Therefore, the test is inconclusive.")
```

In it the parameters of the function local_extrema are the function and a list of the variables respectively. We find the gradient of the function (6), solve the system $\nabla f(\mathbf{x}) = \mathbf{0}$ (9), find the Hessian (15) and for each critical point, we find the eigenvalues and use Theorem 1 to decide what the type of the critical point is (18-30). During that process, we print every intermediate result.

We have preferred to use Theorem 1, due to its ease of coding. We believe that it will not cause any confusion if one decides to use Theorem 2. One would still be able to verify most of their calculations, leaving only some uncertainty about the leading principal minors. That would not be a major issue due to the code classifying the critical points and thus hinting at what the signs of the leading principal minors need to be.

Using this code to get a detailed solution to a local extrema problem can greatly aid the student in both understanding the process of solution and checking for possible miscalculations they may have made. We will show its utility in the following examples.

3 Examples

Example 1. [3] Find the critical points of $f(x,y) = \arctan(x^3 + 2y^3 - 4xy)$, classify them and find the value of the function at those points.

One way to start the code would be to place the variables in a list manually, as is shown below.

```
x, y = sym.symbols("x", real = True), sym.symbols("y", real = True)
f = sym.atan(x**3 + 2*y**3 - 4*x*y)
local_extrema(f,[x,y])
```

The result returned is
$f =$
$\operatorname{atan}\left(x^3 - 4xy + 2y^3\right)$
Gradient of $f =$
$\left(\frac{3x^2-4y}{\left(x^3-4xy+2y^3\right)^2+1}, \frac{-4x+6y^2}{\left(x^3-4xy+2y^3\right)^2+1}\right)$
Critical Points : $x : 0, y : 0, x : 2*2**(2/3)/3, y : 2*2**(1/3)/3,$
Hessian of $f =$

$$\begin{bmatrix} \frac{6x}{\left(x^3-4xy+2y^3\right)^2+1} - \frac{\left(3x^2-4y\right)\left(6x^2-8y\right)\left(x^3-4xy+2y^3\right)}{\left(\left(x^3-4xy+2y^3\right)^2+1\right)^2} & -\frac{\left(-8x+12y^2\right)\left(3x^2-4y\right)\left(x^3-4xy+2y^3\right)}{\left(\left(x^3-4xy+2y^3\right)^2+1\right)^2} - \frac{4}{\left(x^3-4xy+2y^3\right)^2+1} \\ -\frac{\left(-8x+12y^2\right)\left(3x^2-4y\right)\left(x^3-4xy+2y^3\right)}{\left(\left(x^3-4xy+2y^3\right)^2+1\right)^2} - \frac{4}{\left(x^3-4xy+2y^3\right)^2+1} & \frac{12y}{\left(x^3-4xy+2y^3\right)^2+1} - \frac{\left(-8x+12y^2\right)\left(-4x+6y^2\right)\left(x^3-4xy+2y^3\right)}{\left(\left(x^3-4xy+2y^3\right)^2+1\right)^2} \end{bmatrix}$$

Test for $\{x : 0, y : 0\}$.
$m = 4, l = -4$
The smallest eigenvalue $l = -4 < 0,$ whereas the biggest one $m = 4 > 0,$ therefore we have a saddle point $f\_\text{saddle} = 0.$
Test for $\{x : 2*2**(2/3)/3, y : 2*2**(1/3)/3\}$.
$m = 5.25139940381636, l = 1.58072896499492$
The smallest eigenvalue $l = 1.58072896499492 > 0,$ therefore we have a local minimum $f\_\text{min} = -\operatorname{atan}(32/27).$

As we can see, all of the necessary derivatives are computed and the critical points are analysed for potential extrema.

Example 2. Find the critical points of $f(x,y,z) = z\ln(z) - z - z\ln(xy) + xy + x^2 + 2y^2 - 4x - 2y$, classify them and find the value of the function at those points.

With the addition of more variables, writing them out manually can become tedious. That is why we present a second, more automatic way to use the function local_extrema:

```
n = 3
x = [sym.symbols(f"x{i}",real = True) for i in range(1,n+1)]
f = x[2] * sym.log(x[2]) - x[2] - x[2]*sym.log(x[0]*x[1]) + x[0]*x[1] +
    x[0]**2 + 2*x[1]**2-4*x[0]-2*x[1]
local_extrema(f,x)
```

This way we only need to specify how many variables we have and use list comprehension to fill out the necessary list of variables. We can them work with the list directly in order to define f. The result is
$f =$
$x_1^2 + x_1x_2 - 4x_1 + 2x_2^2 - 2x_2 + x_3\log(x_3) - x_3\log(x_1x_2) - x_3$
Gradient of $f =$
$\left(2x_1 + x_2 - 4 - \frac{x_3}{x_1}, x_1 + 4x_2 - 2 - \frac{x_3}{x_2}, \log(x_3) - \log(x_1x_2)\right)$
Critical Points : $\{x1 : 2, x2 : 1/2, x3 : 1\},$
Hessian of $f =$
$$\begin{bmatrix} 2 + \frac{x_3}{x_1^2} & 1 & -\frac{1}{x_1} \\ 1 & 4 + \frac{x_3}{x_2^2} & -\frac{1}{x_2} \\ -\frac{1}{x_1} & -\frac{1}{x_2} & \frac{1}{x_3} \end{bmatrix}$$

Test for $\{x1 : 2, x2 : 1/2, x3 : 1\}$.
$m = 8.71666734232966, l = 0.438015839800740$
The smallest eigenvalue $l = 0.438015839800740 > 0$, therefore we have a local minimum $f\_min = -9/2$.

The only two downsides to this method is that one does not use the usual labeling of the variables as x, y and z and that there is a slight inconsistency with the way the variables x_i are visualized.

Acknowledgments

The research is partially financed by the European Union-NextGenerationEU, through the National Recovery and Resilience Plan of the Republic of Bulgaria, project DUECOS BG-RRP-2.004-0001-C01.

REFERENCES:

[1] Kuttler, K., Calculus of One and Many Variables. Independently published (2021).

[2] Mihovski, S., Mollov, T., Linear Algebra. University Press "Paisii Hilendarski" (2008).

[3] Ashlock, D., Fast Start Advanced Calculus. Morgan & Claypool Publishers (2019).

[4] Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M. J., Terrel, A. R., Roučka, Š., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., Scopatz, A. SymPy: symbolic computing in Python. PeerJ Computer Science. **3**, e103, 2017.
DOI: https://doi.org/10.7717/peerj-cs.103.