AI-Enhanced Performance Evaluation of Python, MATLAB, and Scilab for Solving Nonlinear Systems of Equations: A Comparative Study Using the Broyden Method

Isaac Azure, Japheth Kodua Wiredu, Anas Musah, Eric Akolgo

Scientific Research Publishing

American Journal of Computational Mathematics, 2023, 13, 644-677
https://www.scirp.org/journal/ajcm
ISSN Online: 2161-1211
ISSN Print: 2161-1203

AI-Enhanced Performance Evaluation of Python, MATLAB, and Scilab for Solving Nonlinear Systems of Equations: A Comparative Study Using the Broyden Method

Isaac Azure*, Japheth Kodua Wiredu, Anas Musah, Eric Akolgo

Department of Computer Science, Regentropfen College of Applied Sciences, Bolgatanga, Ghana
Email: *azureike@yahoo.com, wiredujapheth130@gmail.com, bunmalik@gmail.com, eriktita57@gmail.com

How to cite this paper: Azure, I., Wiredu, J.K., Musah, A. and Akolgo, E. (2023) AI-Enhanced Performance Evaluation of Python, MATLAB, and Scilab for Solving Nonlinear Systems of Equations: A Comparative Study Using the Broyden Method. *American Journal of Computational Mathematics*, **13**, 644-677.
https://doi.org/10.4236/ajcm.2023.134036

Received: September 30, 2023
Accepted: December 25, 2023
Published: December 28, 2023

Abstract

This research extensively evaluates three leading mathematical software packages: Python, MATLAB, and Scilab, in the context of solving nonlinear systems of equations with five unknown variables. The study's core objectives include comparing software performance using standardized benchmarks, employing key performance metrics for quantitative assessment, and examining the influence of varying hardware specifications on software efficiency across HP ProBook, HP EliteBook, Dell Inspiron, and Dell Latitude laptops. Results from this investigation reveal insights into the capabilities of these software tools in diverse computing environments. On the HP ProBook, Python consistently outperforms MATLAB in terms of computational time. Python also exhibits a lower robustness index for problems 3 and 5 but matches or surpasses MATLAB for problem 1, for some initial guess values. In contrast, on the HP EliteBook, MATLAB consistently exhibits shorter computational times than Python across all benchmark problems. However, Python maintains a lower robustness index for most problems, except for problem 3, where MATLAB performs better. A notable challenge is Python's failure to converge for problem 4 with certain initial guess values, while MATLAB succeeds in producing results. Analysis on the Dell Inspiron reveals a split in strengths. Python demonstrates superior computational efficiency for some problems, while MATLAB excels in handling others. This pattern extends to the robustness index, with Python showing lower values for some problems, and MATLAB achieving the lowest indices for other problems. In conclusion, this research offers valuable insights into the comparative performance of Python, MATLAB, and Scilab in solving nonlinear systems of equations. It underscores the importance of considering both software and hardware specifica-

tions in real-world applications. The choice between Python and MATLAB can yield distinct advantages depending on the specific problem and computational environment, providing guidance for researchers and practitioners in selecting tools for their unique challenges.

Keywords

System of Nonlinear Equations, Broyden Method, Robustness Index, Artificial Intelligence (AI), MATLAB, SCILAB, Python

1. Introduction

In the realm of computational mathematics and scientific computing, the choice of software can profoundly impact the efficiency and accuracy of numerical solutions. This research represents a significant follow-up to the pioneering work of Isaac Azure, aimed at delving deeper into the performance attributes of three prominent mathematical software packages: Python, MATLAB, and Scilab. These software tools serve as cornerstones in the arsenal of mathematicians, engineers, and scientists, enabling them to tackle complex mathematical problems and simulations with relative ease [1] [2] [3] [4].

The core objective of this study was to conduct a comprehensive assessment of these software packages' capabilities when confronted with the formidable challenge of solving nonlinear systems of equations, each comprising five unknown variables. This is a critical endeavor, as nonlinear systems frequently arise in a myriad of scientific and engineering disciplines, necessitating efficient and reliable numerical techniques for their resolution [5] [6] [7] [8].

A general mathematical representation of a system of nonlinear equations involving "n" variables can be expressed as follows:

Let "x" represent the vector of variables: $x=\left(x_1,x_2,\cdots,x_n\right)$.

The system of nonlinear equations can then be written as:

$$\begin{aligned} f_1(x)&=0\\ f_2(x)&=0\\ &\vdots\\ f_m(x)&=0 \end{aligned} \tag{1}$$

Here, "m" represents the number of equations in the system, and each $f_i(x)$ is a nonlinear function of the variables $x_1,x_2,\cdots,x_n$ that must equal zero. In a more compact vector form, the system can be represented as:

$$F(x)=\begin{bmatrix} f_1(x)\\ f_2(x)\\ \vdots\\ f_m(x)\end{bmatrix}=\begin{bmatrix}0\\0\\\vdots\\0\end{bmatrix} \tag{2}$$

The objective of solving such a system is to find a vector "x" that simulta-

neously satisfies all "m" equations $F(x)=0$. This often involves iterative numerical methods, as analytical solutions are typically not available for arbitrary nonlinear systems [9] [10] [11] [12].

1.1. The Broyden Method

To navigate this intricate terrain, the Broyden method, a widely recognized numerical approach, was chosen as the numerical workhorse for our investigations. The specific focus on nonlinear systems and the utilization of a consistent numerical method provides a controlled and rigorous framework for the comparative analysis of Python, MATLAB, and Scilab [13].

The Broyden method, also known as the Broyden–Fletcher–Goldfarb–Shanno (BFGS) update method, is an iterative numerical technique used to solve systems of nonlinear equations. It is named after its developers, Charles W. Broyden and Roger Fletcher, and it is an extension of the more well-known Broyden method for solving nonlinear systems. The BFGS variant is primarily used for solving unconstrained nonlinear optimization problems, while the original Broyden method is designed for solving systems of nonlinear equations [14].

The Broyden method was developed as an alternative to traditional Newton's method for solving nonlinear equations. Newton's method involves iteratively linearizing a nonlinear system of equations and solving the linearized system at each iteration. However, Newton's method can be computationally expensive, especially when dealing with large systems, because it requires the computation of the Jacobian matrix (a matrix of partial derivatives) at each iteration [15] [16].

The Broyden method aims to overcome this computational burden by approximating the Jacobian matrix with each iteration, rather than recomputing it from scratch. It does so by updating an initial approximation of the Jacobian matrix using information from previous iterations. This update makes the method more efficient and less computationally demanding than Newton's method [17].

The Broyden method is adopted as the numerical technique for solving the nonlinear systems in each software package, providing a consistent approach for performance assessment. According to Charles Broyden, (1965), two methods can be used to find the approximate solution for nonlinear systems of equations as reported in [18].

The first method gives an approximate matrix for B_k using the following assumption;

B_k must satisfy the secant equation

$$B_k s^{(k)} = y^{(k)} \tag{3}$$

where

$$s^{(k)} = x^{(k)} - x^{(k-1)} \tag{4}$$

and

$$y^{(k)} = F\left(x^{(k)}\right) - F\left(x^{(k-1)}\right) \tag{5}$$

However, Broyden's method involves the computation of B_k^{-1} and not B_k, this brings our attention to the next theorem.

THEOREM: (Sherman-Morrison Formula) If B is a nonsingular matrix and x and y are vectors, then $B+xy^t$ is nonsingular provided that $y^tA^{-1}x \neq -1$ and

$$\left(B+xy^t\right)^{-1} = B^{-1} - \frac{B^{-1}xy^tB^{-1}}{1+y^tB^{-1}x} \tag{6}$$

The theorem above is a matrix inverse formula. It allows B_k^{-1} to be computed directly using B_{k-1}^{-1}, rather than computing B_k and then its inverse at each iteration. Hence, using the theorem and setting $B = B_{k+1}$, $x = \frac{y_i - B_{k-1}s_k}{\|s_k\|_2^2}$, and $y = s_k$, as well as using B_k as defined above we have that

$$B_k^{-1} = \left(B_{k-1} + \frac{y_k - B_{k-1}s_k}{\|s_k\|_2^2}s_k^t\right)^{-1} \tag{7}$$

$$B_k^{-1} = B_{k-1}^{-1} - \frac{B_{k-1}^{-1}\left(B_{k-1} + \frac{y_k - B_{k-1}s_k}{\|s_k\|_2^2}s_k^t\right)B_{k-1}^{-1}}{1+s_k^tB_{k-1}^{-1}\left(\frac{y_k - B_{k-1}s_k}{\|s_k\|_2^2}\right)} \tag{8}$$

$$B_k^{-1} = B_{k-1}^{-1} - \frac{\left(B_{k-1}^{-1}y_k - s_i\right)s_k^tB_{k-1}^{-1}}{\|s_k\|_2^2 + s_k^tB_{k-1}^{-1}y_k - \|s_k\|_2^2} \tag{9}$$

Hence, we get

$$B_k^{-1} = B_{k-1}^{-1} + \frac{\left(s_k - B_{k-1}^{-1}y_k\right)s_k^tB_{k-1}^{-1}}{s_k^tB_{k-1}^{-1}y_k} \tag{10}$$

From the assumption above, Broyden's method is defined as

$$x^{(k+1)} = x^{(k)} - B_k^{-1}F\left(x^{(k)}\right) \tag{11}$$

where B_k^{-1} is computed using Equation (10).

1.2. Related Works

Nonlinear equations, unlike their linear counterparts, feature at least one nonlinear term, making them considerably more challenging to solve. In tackling these intricate equations, numerical methods come to the rescue, often relying on an initial guess [19].

In a closely related study, five numerical techniques for resolving nonlinear equations were investigated after obtaining their solutions manually. The Bisection method, Newton-Raphson method, Regula Falsi method, Secant method, and Fixed-Point Iteration method were all subject to comparison. Researchers developed manual computational algorithms for each approach and employed them to manually find a root using a TI-Inspire calculator. All methods eventually converged to an exact solution. However, the Bisection method reached

convergence by the 14th iteration, the Fixed-Point Iterative Method by the 7th iteration, the Secant method by the 5th iteration, and both the Regula Falsi and Newton Raphson methods by the 2nd iteration [20].

In another pertinent study, the challenge of finding roots for nonlinear equations, arising frequently in practical applications across science and engineering, was extensively explored. The process of locating a root is referred to as root-finding, with the value of "x" that satisfies $f(x)=0$ termed a root of $f(x)=0$. This research meticulously compared the convergence rates of two prevalent root-finding methods: Bisection and Newton-Raphson. MATLAB software was harnessed to locate the root of a specific function and juxtapose the outcomes of these two methods. The study's conclusion favored Newton's approach, which outperformed the Bisection method [20].

Throughout these aforementioned studies, MATLAB software took center stage in estimating the roots of nonlinear equations. In a related comparative analysis, Python emerged as the frontrunner due to its remarkable efficiency and precision. Python achieved fitting approximations with the fewest iterations and minimal computational time. Notably, the research underscored the Newton-Raphson method's robustness, as it consistently converged efficiently with minimal iteration counts across various benchmark problems. This underscored the method's superiority in terms of efficiency and reliability, especially when dealing with complex nonlinear equations [21].

This research is a continuation of the work aimed at identifying which mathematical software package performs better numerical analysis. It however, set out to discern the disparities among three mathematical software packages: Python, Scilab, and MATLAB, particularly in the context of solving a system of nonlinear equations using the Broyden method on different computer environments [3].

2. Objectives of the Study

The objectives of this study are:

1) To assess and compare the performance of Python, MATLAB, and Scilab in solving nonlinear systems of equations with five unknown variables.

2) To utilize a set of five standardized benchmark problems, featuring nonlinear equations, to provide a consistent platform for comparative analysis.

3) To employ key performance metrics, including computational time, convergence iterations, and robustness index, to quantitatively evaluate software performance.

4) To investigate the influence of varying computer hardware specifications on software performance, emphasizing the importance of hardware considerations in software selection and real-world applications.

3. Methodology

The methodology employed in this research builds upon the seminal work of Isaac Azure, aiming to rigorously assess the performance of three prominent mathe-

matical software packages: Python, MATLAB, and Scilab. The research seeks to comprehensively investigate their capabilities in solving nonlinear systems of equations featuring five unknown variables. Key metrics, including computational time, the number of iterations for convergence, and the robustness index, are utilized to differentiate the software packages. Furthermore, the study explores the influence of varying computer hardware specifications on computational performance and solution robustness, ultimately providing valuable insights for software selection and real-world applications.

3.1. Benchmark Problems Selection

Five benchmark problems comprising of a system of nonlinear equations with five unknowns were chosen to serve as test cases, ensuring standardized evaluation across the software packages.

Problem 1

$$\begin{aligned} e^{x_1} + \sin(x_2) - x_3^2 + \cos(x_4) - x_5 &= 2 \\ x_1 + \ln(x_2) + x_3 + \tan(x_4) - x_5^2 &= 1 \\ x_1^2 - e^{x_2} + x_3 + \sin(x_4) + \cos(x_5) &= 3 \\ \cos(x_1) - x_2 + x_3^3 + \ln(x_4) + \tan(x_5) &= 0 \\ \tan(x_1) - x_2 + \sin(x_3) - \cos(x_4) + e^{x_5} &= 1 \end{aligned} \tag{12}$$

Problem 2

$$\begin{aligned} \ln(x_1) + \sin(x_2) - x_3^2 + \cos(x_4) - x_5 &= 1.5 \\ x_1^2 + \ln(x_2) + x_3 + \tan(x_4) - x_5^2 &= 2 \\ e^{x_1} - e^{x_2} + x_3 + \sin(x_4) + \cos(x_5) &= 3 \\ \cos(x_1) - x_2 + x_3^2 + \ln(x_4) + \tan(x_5) &= 0 \\ \tan(x_1) - x_2 + \sin(x_3) - \cos(x_4) + e^{x_5} &= 1 \end{aligned} \tag{13}$$

Problem 3

$$\begin{aligned} e^{x_1} + \sin(x_2) - x_3^2 + \cos(x_4) - x_5 &= 2 \\ x_1 + x_2^2 + x_3 + \tan(x_4) - \ln(x_5) &= 1 \\ x_1^2 - e^{x_2} + x_3 + \sin(x_4) + \cos(x_5) &= 3 \\ \cos(x_1) - x_2 + x_3^2 + \ln(x_4) + \tan(x_5) &= 0 \\ \tan(x_1) - x_2 + \sin(x_3) - \cos(x_4) + e^{x_5} &= 1 \end{aligned} \tag{14}$$

Problem 4

$$\begin{aligned} \ln(x_1) + \sin(x_2) - x_3^2 + \cos(x_4) - x_5 &= 1.5 \\ x_1^2 + \ln(x_2) + x_3 + \tan(x_4) - x_5^2 &= 2 \\ e^{x_1} - e^{x_2} + x_3 + \sin(x_4) + \cos(x_5) &= 3 \end{aligned} \tag{15}$$

$$\cos(x_1) - x_2 + x_3^2 + \ln(x_4) + \tan(x_5) = 0$$

$$\tan(x_1) - x_2 + \sin(x_3) - \cos(x_4) + e^{x_5} = 1$$

Problem 5

$$\begin{aligned} e^{x_1} + \sin(x_2) - x_3^2 + \cos(x_4) - x_5 &= 2 \\ x_1 + x_2^2 + x_3 + \tan(x_4) - \ln(x_5) &= 1 \\ x_1^2 - e^{x_2} + x_3 + \sin(x_4) + \cos(x_5) &= 3 \\ \cos(x_1) - x_2 + x_3^2 + \ln(x_4) + \tan(x_5) &= 0 \\ \tan(x_1) - x_2 + \sin(x_3) - \cos(x_4) + e^{x_5} &= 1 \end{aligned} \tag{16}$$

The following initial guess values (Table 1) were used for the estimation of the root of the benchmark problems with the help of the Broyden method.

3.2. Numerical Methodology

Below is the Broyden's method algorithm:

STEP 1: Let $x^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, \cdots, x_n^{(0)}\right)$ be the initial vector given.

STEP 2: Calculate $F\left(x^{(0)}\right)$.

STEP 3: In this step we compute B_0^{-1}. Because we do not have enough information to compute B_0 directly, Broyden's method permits us to let $B_0 = J\left(x^{(0)}\right)$, which implies that $B_0^{-1} = J\left(x^{(0)}\right)^{-1}$.

Table 1. Selected initial guess values for benchmark problems.

PROBLEM	INITIAL GUESS VALUES (IG)	
Problem 1	IG 1:	0.1, 0.2, −0.3, 0.4, 0.5
	IG 2:	0.2, 0.3, −0.4, 0.5, 0.6
	IG 3:	0.2, 0.2, −0.4, 0.4, 0.6
	IG 4:	0.3, 0.4, −0.5, 0.6, 0.7
Problem 2	IG 1:	0.8, 1.2, −0.6, 0.3, 1.7
	IG 2:	0.9, 1.3, −0.7, 0.4, 1.8
	IG 3:	0.9, 1.2, −0.7, 0.3, 1.8
	IG 4:	1.0, 1.4, −0.8, 0.5, 1.9
Problem 3	IG 1:	1.2, 0.6, −0.8, 0.4, 1.5
	IG 2:	1.3, 0.7, −0.9, 0.5, 1.6
	IG 3:	1.3, 0.6, −0.9, 0.4, 1.6
	IG 4:	1.4, 0.8, −1.1, 0.6, 1.8
Problem 4	IG 1:	1.0, 0.8, −0.4, 0.2, 1.2
	IG 2:	1.1, 0.9, −0.5, 0.3, 1.3
	IG 3:	1.1, 0.8, −0.5, 0.2, 1.3
	IG 4:	1.2, 1.0, −0.6, 0.4, 1.4
Problem 5	IG 1:	0.7, 0.9, −0.3, 0.5, 1.3
	IG 2:	0.8, 1.0, −0.4, 0.6, 1.4
	IG 3:	0.8, 0.9, −0.4, 0.5, 1.4
	IG 4:	0.9, 1.1, −0.5, 0.7, 1.5

STEP 4: Calculate $x^{(1)} = x^{(0)} - B_0^{-1} F\left(x^{(0)}\right)$.

STEP 5: Calculate $F\left(x^{(1)}\right)$.

STEP 6: Take $F\left(x^{(0)}\right)$ and $F\left(x^{(1)}\right)$ and calculate $y_1 = F\left(x^{(1)}\right) - F\left(x^{(0)}\right)$. Next, take the first two iterations of $x^{(k)}$ and calculate $s_1 = x^{(1)} - x^{(0)}$.

STEP 7: Calculate $s_k^t B_{k-1}^{-1} y_k$.

STEP 8: Compute $B_1^{-1} = B_0^{-1} + \dfrac{1}{s_1^t B_0^{-1} y_1}\left[\left(s_1 - B_0^{-1} y_1\right) s_1^t B_0^{-1}\right]$.

STEP 9: Take B_1^{-1} that we found in step 8, and calculate $x^{(2)} = x^{(1)} - B_1^{-1} F\left(x^{(1)}\right)$.

STEP 10: Repeat the process until it converges at $\bar{x}$, *i.e.* when $x^{(k)} = x^{(k+1)} = \bar{x}$. This will indicate that we have solved of the system.

Broyden's method as well as all of the Quasi-Newton methods converge superlinear, which means that

$$\lim_{k\to\infty} \frac{\left\|x^{(k+1)} - p\right\|}{\left\|x^{(k)} - p\right\|} = 0 \tag{17}$$

where p is the solution to $F(x) = 0$, and $x^{(k)}$ and $x^{(k+1)}$ are successive approximations to p.

3.3. Performance Metrics

A triad of pivotal performance metrics guided our exploration: computational time, the number of iterations needed for convergence, and the robustness index. These metrics collectively encapsulate the essence of software performance in numerical problem-solving. Computational time offers insights into the efficiency and speed of each software package, iterations unveil the underlying numerical convergence properties, and the robustness index serves as a gauge of solution reliability and stability.

The robustness index is a quantitative measure of how well a given solution satisfies a system of nonlinear equations. The algorithm for the calculation of the robustness index for the system of nonlinear equations:

STEP 1: Define the system of nonlinear equations: Begin with a set of nonlinear equations, typically represented as $F(x) = 0$, where F is a vector-valued function of x, and x is the vector of the unknowns you want to solve.

STEP 2: Find a solution: Use a numerical solver or method (Broyden method) to find a solution x^* that approximately satisfies the system of equations, *i.e.*, $F\left(x^*\right) \approx 0$.

STEP 3: Calculate the Residual Vector: The residual vector, denoted as R, is defined as the vector of the differences between the left-hand side (LHS) and the right-hand side (RHS) of the equations for the solution x^*. Mathematically, it can be expressed as:

$$R\left(x^*\right) = F\left(x^*\right).$$

where:

$R\left(x^*\right)$ is the residual vector.

$F\left(x^*\right)$ is the vector of equations evaluated at the solution x^*.

STEP 4: Calculate the Norm (Magnitude) of the Residual Vector: The 2-norm (Euclidean norm) of the residual vector is computed to measure how far the solution is from satisfying the equations. The 2-norm of a vector υ is calculated as:

$$\|\upsilon\|_2 = \sqrt{\sum_{i=1}^{n} |\upsilon_i|^2}$$

where:

$\|\upsilon\|_2$ is the 2-norm of the vector υ.

υ_i represents the i-th component of vector υ.

n is the dimension of the vector.

In our case, υ is the residual vector $R\left(x^*\right)$.

STEP 5: Display the robustness index: The robustness index is simply the value of the 2-norm of the residual vector $R\left(x^*\right)$. It quantifies how well the solution x^* satisfies the equations. A smaller robustness index indicates a solution that is closer to satisfying the equations. In summary, the robustness index is calculated by finding a solution to the system of nonlinear equations, calculating the residual vector that measures how well the solution satisfies the equations, and then computing the 2-norm (Euclidean norm) of the residual vector to quantify the solution's robustness or closeness to satisfying the equations.

3.4. Experimental Setup

The study involved multiple facets of experimentation, ranging from software configuration to hardware variation. In the case of the software configuration, the 2023 versions of the Python, MATLAB, and Scilab packages were employed for the study. Codes were developed for each benchmark problem using the Broyden method and ensuring that each code looked out for the root of the problems, computational time, number of iterations, and the robustness index.

The second part of the experimental setup considered the hardware variation. Here, four laptops with different specifications were used simultaneously to run the codes (Python, MATLAB, Scilab) developed to solve the benchmark problems. The table below (Table 2) shows the specifications of the laptop computers used for this study:

Table 2. Specifications of Laptop devices.

No	DEVICE NAME	PROCESSOR	INSTALLED RAM	SYSTEM TYPE	WINDOWS EDITION
1	HP PROBOOK	Intel® Core™ i5-8265U CPU@ 1.60GHZ 1.80GHZ	8.00 GB (7.78 GB usable)	64-bit Operating System, x64-based processor	Windows 10
2	HP ELITEBOOK FOLIO 9470m	Intel® Core™ i5-3427U CPU@ 1.80GHZ 2.30GHZ	8.00 GB (7.87 GB usable)	64-bit Operating System, x64-based processor	Windows 10
3	DELL INSPIRON	Intel® Core™ i7-8550U CPU@ 1.80GHZ 1.99GHZ	12.00 GB (11.9 GB usable)	64-bit Operating System, x64-based processor	Windows 11
4	DELL LATITUDE	Intel® Core™ i7-4310U CPU@ 2.00GHZ 2.60GHZ	8.00 GB	64-bit Operating System, x64-based processor	Windows 10

4. Results and Discussion

Pursuant to the fourth objective of this investigation, we subjected the codes crafted for the three mathematical software packages to rigorous testing on a spectrum of four distinct laptop computers, each characterized by unique specifications, as detailed in Table 2. The overarching aim was to evaluate the performance of Python, MATLAB, and Scilab in root-finding for five designated benchmark problems. This evaluation encompassed the computation time, iteration count, and robustness index, all adjudicated through the lens of the Broyden method. Notably, these software packages were examined across an eclectic array of computing environments, namely the HP Probook, HP Elitebook, Dell Inspiron, and Dell Latitude laptops.

It is worth highlighting that within this triad of mathematical tools, Scilab faced convergence challenges for all five benchmark problems across the four computers employed in this study. Consequently, our dataset predominantly relies on the outcomes produced by MATLAB and Python software. Each table below is accompanied by a graphical representation, facilitating a comprehensive juxtaposition of computational times and robustness indices for Python and MATLAB across diverse computing environments.

Using the initial guess values in Table 1 above, the solutions of the bench-mark problems using the HP Probook laptop is shown in Table 3 below are graphs showing each problem was solved using the software.

Table 3. Results of HP Probook Laptop.

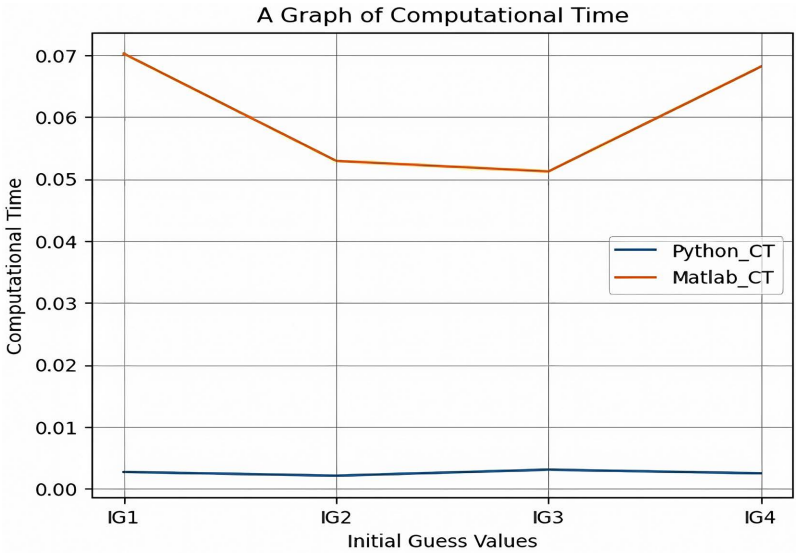
PROBLEM	PACKAGE	ROOT					ITERATIONS	CT	RI
		x_1	x_2	x_3	x_4	x_5			
	PYTHON								
	IG 1	0.9703856	0.33640073	1.78248243	0.70765518	−1.25063674	78	0.002707190	0.97113728
	IG 2	0.97150346	0.33724451	1.78485727	0.7072669	−1.25077743	83	0.00210545	0.97110238
	IG 3	0.96928977	0.33656571	1.78236457	0.7149359	−1.2518972	115	0.0030783653	0.97105340
	IG 4	0.8167544	−0.35966987	1.32197719	0.16833372	−0.71637541	84	0.0024866318	1.2295925548
1	**MATLAB**								
	IG 1	0.9716	0.3363	1.7865	0.7119	−1.2533	66	0.0702	0.9710
	IG 2	0.9716	0.3363	1.7865	0.7119	−1.2533	83	0.0529	0.9710
	IG 3	0.9959	0.2987	1.7370	0.1282	−0.8380	67	0.0512	1.0835
	IG 4	0.9716	0.3363	1.7865	1.7119	−1.2533	76	0.0681	0.9710
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
	IG 1	0.7410159	0.616552	−0.68722343	1.40086399	1.83923133	82	0.0026665	6.9985642
2	IG 2	0.73498977	0.62935176	−0.67192174	1.40004509	1.84389973	76	0.0024065518	6.99792766
	IG 3	0.72778208	0.61529578	−0.67099376	1.40235684	1.84342885	62	0.0017950415	6.9979516
	IG 4	0.72746388	0.61529185	−0.70037927	1.40542718	1.8440568	67	0.001944973	6.9996142

Continued

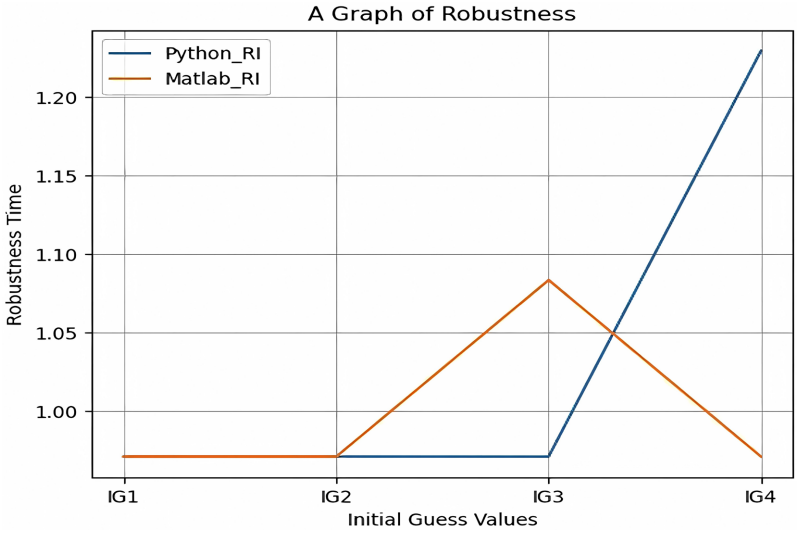
	MATLAB								
	IG 1	0.7326	0.6239	−0.6653	1.4011	1.8434	228	0.1732	6.9978
	IG 2	0.7326	0.6239	−0.6653	1.4011	1.8434	263	0.1744	6.9978
	IG 3	0.7326	0.6239	−0.6653	1.4011	1.8434	277	0.2148	6.9978
	IG 4	0.7326	0.6239	−0.6653	1.4011	1.8434	343	0.2690	6.9978
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
	IG 1	0.64170722	−0.05649459	0.65815341	−0.0862085	0.58320854	148	0.00356941	2.615594
	IG 2	0.69352355	−0.27318825	0.2337675	0.25145785	0.46148114	80	0.00199568	2.38164651
	IG 3	0.66576994	−0.27633288	0.24413165	0.23223115	0.45262216	87	0.002296090	2.37832328
	IG 4	0.77618451	−0.31053416	0.04072139	0.24611656	0.52286966	86	0.002186186	2.41270966
3	**MATLAB**								
	IG 1	0.6353	−0.2561	0.3129	0.2223	0.4475	58	0.0470	2.3772
	IG 2	−1.4359	0.3031	0.4137	1.1899	2.1527	400	0.2333	3.9702
	IG 3	−1.4359	0.3031	0.4137	1.1899	2.1527	400	0.2441	3.9702
	IG 4	0.6621	0.0829	−0.7891	0.9747	1.8349	400	0.0726	6.8038
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
	IG 1	0.99699399	0.4663226	−0.11985545	1.10630865	−0.37413413	147	0.0027879	0.4174182
	IG 2	0.79460544	−0.00306158	1.10882098	0.17891018	0.00779873	102	0.003098425	1.44879087
	IG 3	−0.21843663	−0.41870705	0.86175569	0.88075046	−1.07887289	135	0.00465234	1.6819834
	IG 4	1.03781398	0.88471088	0.01948704	0.78311119	0.67483508	77	0.0020654129	1.752369535
4	**MATLAB**								
	IG 1	0.9518	0.4318	−0.0050	1.1308	−0.4562	196	0.1592	0.4052
	IG 2	0.9518	0.4318	−0.0050	1.1308	−0.4562	276	0.1527	0.4052
	IG 3	0.9518	0.4318	−0.0050	1.1308	−0.4562	225	0.1425	0.4052
	IG 4	0.9518	0.4318	−0.0050	1.308	−0.4562	184	0.1123	0.4052
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
	IG 1	7.93165560e−01	−8.55523324e−04	1.12190615e+00	1.66453597e−01	2.19405901e−03	107	0.002276010	1.44110698
	IG 2	0.79460544	−0.00306158	1.10882098	0.17891018	0.00779873	102	0.003278701	1.44879087
	IG 3	−0.21843663	−0.41870705	0.86175569	0.88075046	−1.07887289	135	0.00465234	1.6819834
	IG 4	1.82494083e+00	2.35951827e−02	1.82997012e+00	−1.19703799e−07	1.49325421e+00	162	0.00445223	1.352219796
5	**MATLAB**								
	IG 1	0.6353	−0.2561	0.3129	0.2223	0.4475	94	0.0504	2.3772
	IG 2	0.6353	−0.2561	0.3129	0.2223	0.4475	91	0.0507	2.3772
	IG 3	0.6353	−0.2561	0.3129	0.2223	0.4475	91	0.0448	2.3772
	IG 4	0.6353	−0.2561	0.3129	0.2223	0.4475	86	0.0473	2.3772
	SCILAB	-	-	-	-	-	-	-	-

The collected data from running the codes on an HP Probook laptop reveals noteworthy insights. Python consistently exhibited significantly shorter computational times across all five benchmark problems when compared to MATLAB and this can be seen in Figures 1(a)-5(a). Specifically, Python's computational time ranged from 0.0017950 seconds as the lowest to 0.00465234 seconds as the highest, whereas MATLAB's computational time spanned from 0.0448 seconds (minimum) to 0.2690 seconds (maximum).

Regarding the robustness index, Python outperformed MATLAB by achieving a lower index in problems 3 and 5. Notably, for problem 1, Python achieved an identical index as MATLAB for initial guess values 1 and 2. However, Python surpassed MATLAB for initial guess 3, while MATLAB exhibited better performance for initial guess values 4 (Figures 1(b)-5(b)).

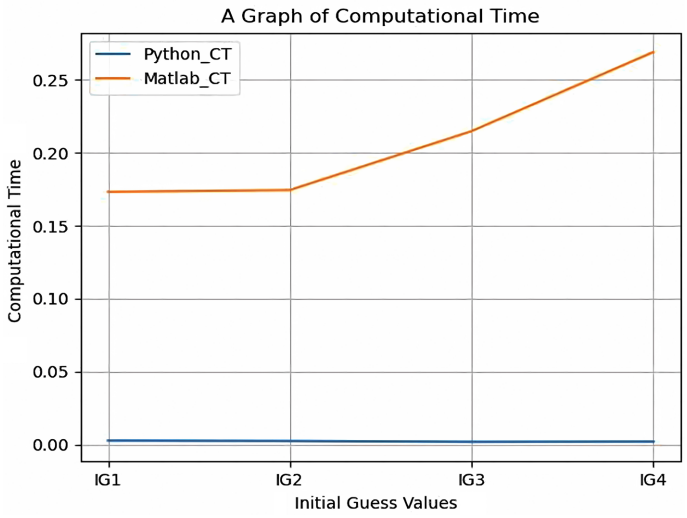


(a)

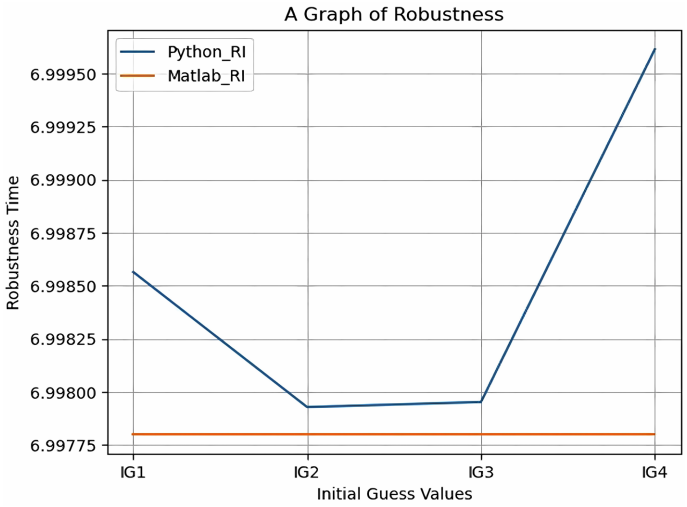


(b)

Figure 1. (a) CT against IG for Problem 1; (b) RI against IG for Problem 1.

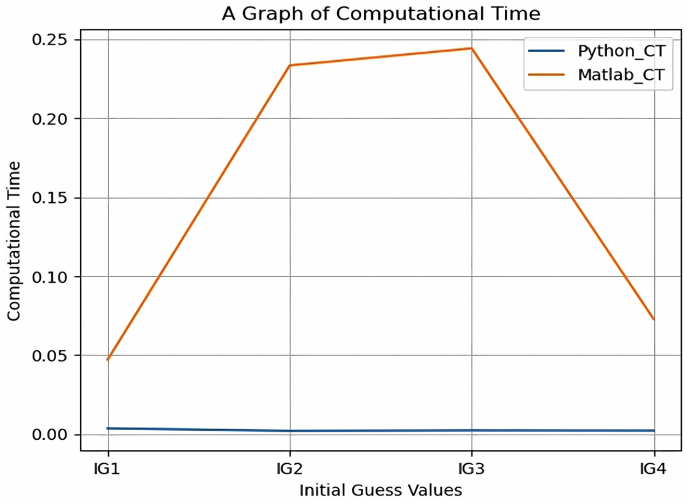


(a)

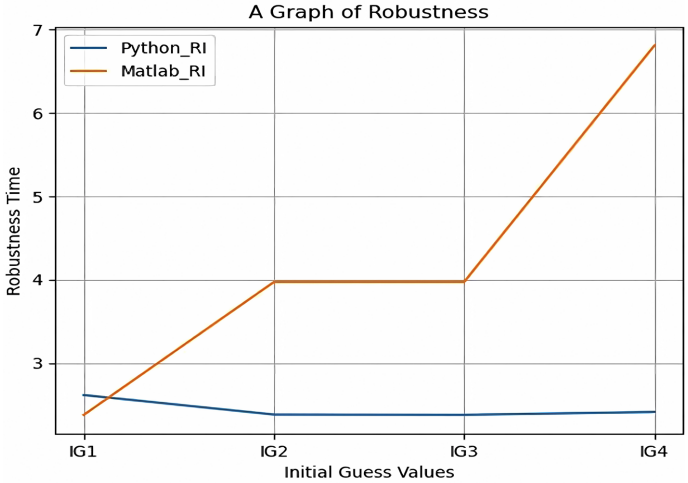


(b)

Figure 2. (a) CT against IG for Problem 2; (b) RI against IG for Problem 2.

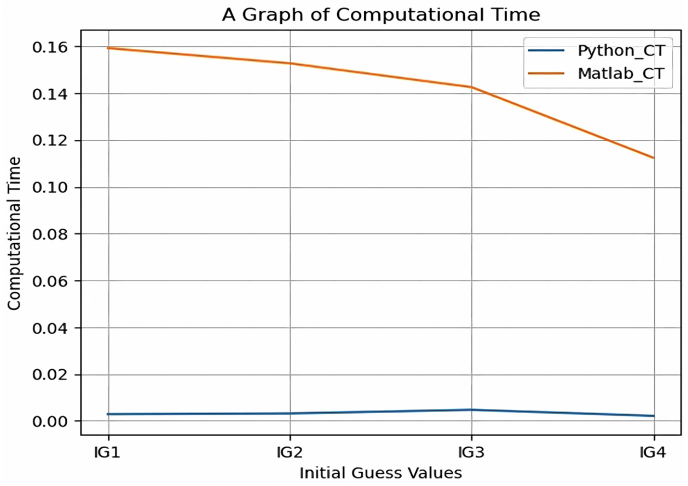


(a)

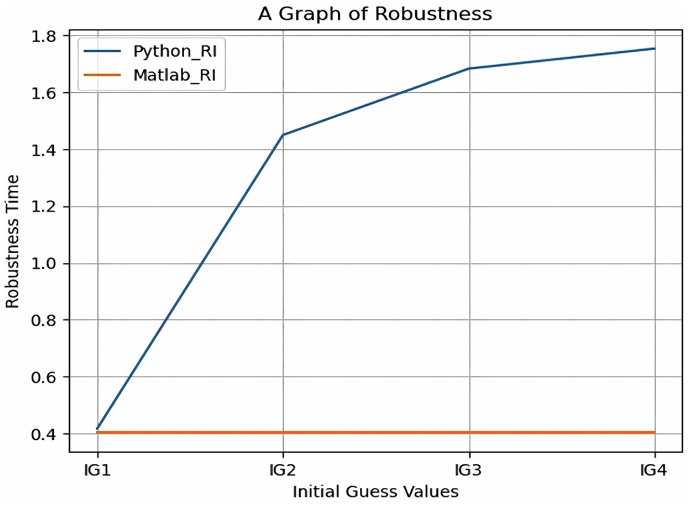


(b)

Figure 3. (a) CT against IG for Problem 3; (b) RI against IG for Problem 3.

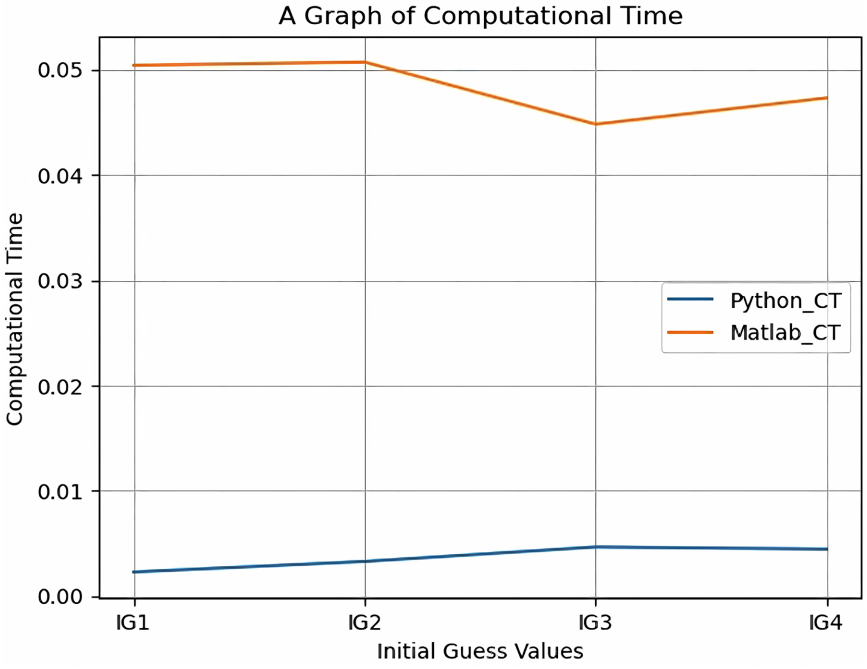


(a)

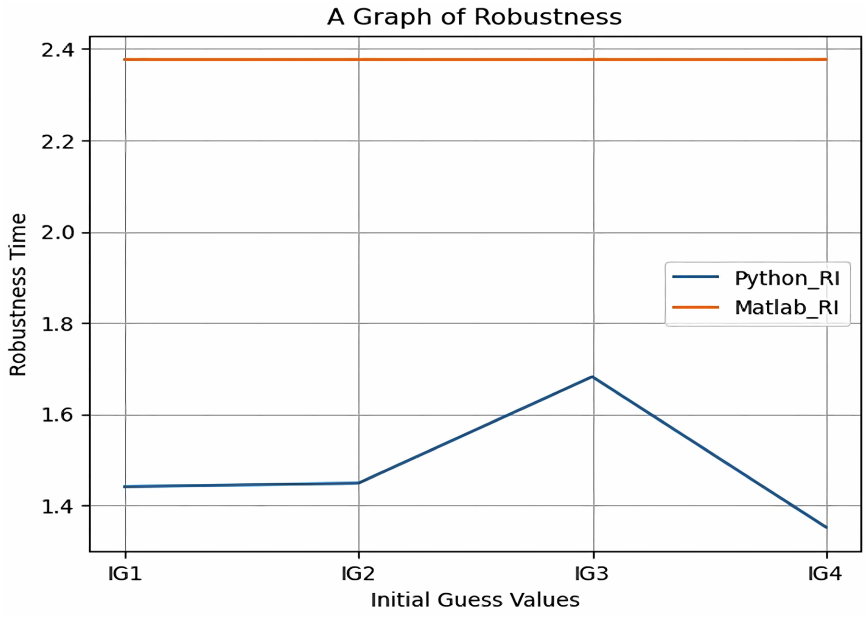


(b)

Figure 4. (a) CT against IG for Problem 4; (b) RI against IG for Problem 4.



(a)



(b)

Figure 5. (a) CT against IG for Problem 5; (b) RI against IG for Problem 5.

Table 4 is a summary of results obtained from HP Elitebook Laptop.

When examining the data collected on the HP Elitebook laptopas shown in **Table 4**, intriguing disparities emerged, presenting a stark contrast to the findings on the Probook HP device. Unlike the Probook HP laptop, MATLAB consistently exhibited notably shorter computational times than Python for all four benchmark problems investigated in this study, resulting in significant differences between the two software platforms.

Table 4. Results of HP Elitebook Laptop.

PROBLEM	PACKAGE	ROOT					ITERATIONS	CT	RI
		x_1	x_2	x_3	x_4	x_5			
	PYTHON								
	IG 1	0.97039	0.336401	1.78248	0.70766	−1.2506	78	0.00359	0.971137
	IG 2	0.97150	0.337244	1.78486	0.70727	−1.2507	83	0.00399	0.971102
	IG 3	0.96929	0.336566	1.78236	0.71494	−1.2518	115	0.00529	0.971053
	IG 4	0.81675	−0.35967	1.32198	0.16834	−0.7163	84	0.00470	1.229526
1	**MATLAB**								
	IG 1	0.9959	0.2987	1.7370	0.1282	−0.8380	66	0.0064	1.083549
	IG 2	0.9716	0.3363	1.7865	0.7119	−1.2533	83	0.0013	0.970984
	IG 3	0.9959	0.2987	1.7370	0.1282	−0.8380	67	0.0011	1.083549
	IG 4	0.9716	0.3363	1.7865	1.7119	−1.2533	76	0.0011	0.970984
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
	IG 1	0.7410	0.6166	−0.6872	1.4009	1.8392	82	0.00432	6.998564
	IG 2	0.7349	0.6294	−0.6719	1.0005	1.8439	76	0.00404	6.997928
	IG 3	0.7278	0.6153	−0.6709	1.4025	1.8434	62	0.00439	6.997952
	IG 4	0.7275	0.6153	−0.7004	1.4054	1.8441	67	0.00416	6.999614
2	**MATLAB**								
	IG 1	0.7326	0.6239	−0.6653	1.4011	1.8434	228	0.0059	6.997791
	IG 2	0.7326	0.6239	−0.6653	1.4011	1.8434	263	0.0023	6.997791
	IG 3	0.7326	0.6239	−0.6653	1.4011	1.8434	277	0.0024	6.997791
	IG 4	0.7326	0.6239	−0.6653	1.4011	1.8434	343	0.0028	0.997791
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
	IG 1	0.6561	−0.3227	0.5981	−0.0842	0.6430	151	0.00706	2.663338
	IG 2	0.8312	0.3925	2.4179	0.4889	1.6708	87	0.00401	9.276602
	IG 3	0.6123	0.4125	2.3623	0.7147	1.6543	72	0.00347	10.305454
	IG 4	0.8870	0.5897	−0.4366	2.1678	1.8248	44	0.00238	8.159903
3	**MATLAB**								
	IG 1	0.6353	−0.2561	0.3129	0.2223	0.4475	72	0.0015	2.377167
	IG 2	−1.4358	0.2983	0.4158	1.1896	2.1513	400	0.0033	3.970227
	IG 3	−1.4359	0.2998	0.4148	1.1897	2.1519	400	0.0029	3.970228
	IG 4	0.6595	0.0822	−0.7859	0.9747	1.8351	400	0.0025	6.803822
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
4	IG 1	0.9970	0.4663	−0.1199	1.1063	−0.3741	147	0.00644	0.417418
	IG 2	-	-	-	-	-	-	-	-

Continued

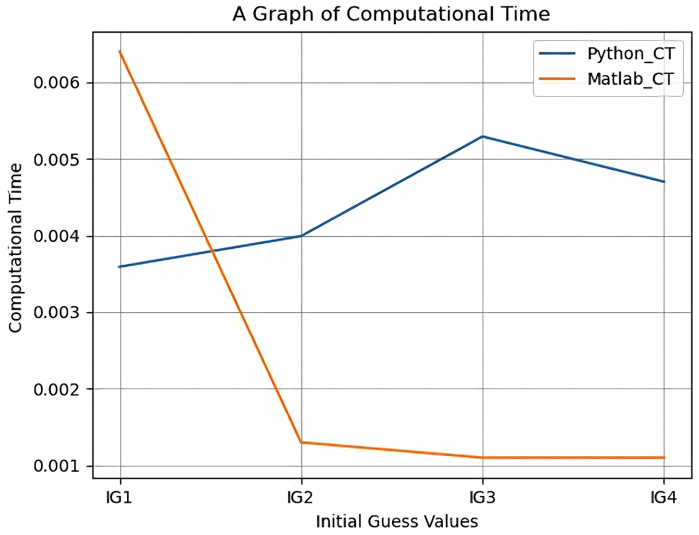
	IG 3	1.0593	0.5232	−0.2939	1.0706	−0.2795	146	0.00652	0.481636
	IG 4	1.0378	0.8847	0.0195	0.7831	0.6748	77	0.00363	1.75236
	MATLAB								
	IG 1	0.9518	0.4318	−0.0050	1.1308	−0.4562	179	0.0059	0.405212
	IG 2	0.9518	0.4318	−0.0050	1.1308	−0.4562	276	0.0026	0.405212
	IG 3	0.9518	0.4318	−0.0050	1.1308	−0.4562	225	0.0021	0.405212
	IG 4	0.9518	0.4318	−0.0050	1.308	−0.4562	184	0.0020	0.405212
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
	IG 1	0.7932	−0.0008	1.1219	0.6646	0.0021	107	0.00540	1.441107
	IG 2	0.7946	−0.0031	1.1088	0.1789	0.0078	102	0.00516	1.448791
	IG 3	0.0492	−0.2693	1.1124	0.9893	−1.1836	121	0.00597	1.614661
	IG 4	1.8249	0.0236	1.8299	−1.1960	1.4933	162	0.00779	1.352203
5	**MATLAB**								
	IG 1	0.6353	−0.2561	0.3129	0.2223	0.4475	94	0.0013	2.377167
	IG 2	0.6353	−0.2561	0.3129	0.2223	0.4475	91	0.0013	2.377167
	IG 3	0.6353	−0.2561	0.3129	0.2223	0.4475	91	0.0013	2.377167
	IG 4	0.6353	−0.2561	0.3129	0.2223	0.4475	86	0.0012	2.377167
	SCILAB	-	-	-	-	-	-	-	-

The analysis of the robustness index data revealed a consistent pattern. Employing the Broyden method, MATLAB consistently yielded a lower robustness index than Python for all benchmark problems, except for problem 5. However, the data obtained from the HP Elitebook laptop unveiled a distinct challenge; Python failed to converge for problem 4 (**Figure 9(a)**) with initial guess values 2, while MATLAB successfully delivered results for the same problem with those initial guess values. In general, MATLAB consistently solved all five bench-mark problems with lower computational time as shown in the figures above (**Figures 6(a)-10(a)**). In a similar way, MATLAB recorded the least robustness index as compared with Python (**Figures 6(b)-10(b)**).

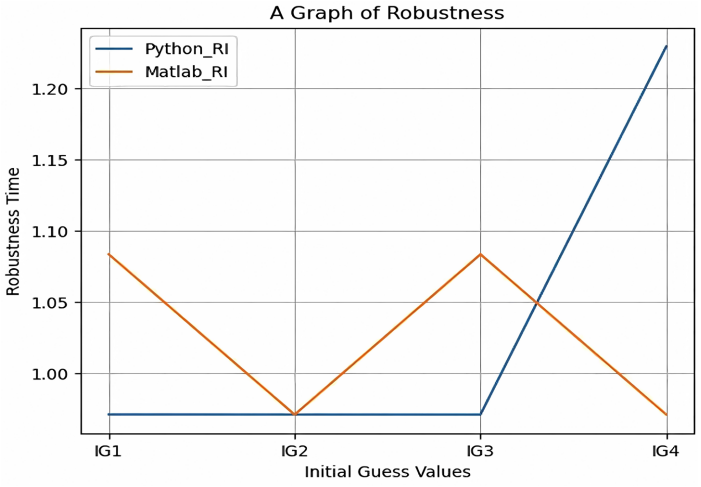
A summary of results obtained from Dell Inspiron Laptop is displayed in **Table 5**.

Analyzing the data acquired from the Dell Inspiron laptop as shown in **Table 5**, intriguing patterns emerged in terms of computational efficiency and robustness. Python outshone MATLAB in terms of computational time, delivering quicker solutions for problems 2, 3, and 4 (**Figures 12(a)-14(a)**). In contrast, MATLAB displayed its computational prowess by surpassing Python in tackling problems 2 and 5 (**Figure 11(a)** and **Figure 15(a)**).

The story was similar when it came to the robustness index. Python demonstrated a significantly smaller robustness index for problems 1, 3, and 5, proving

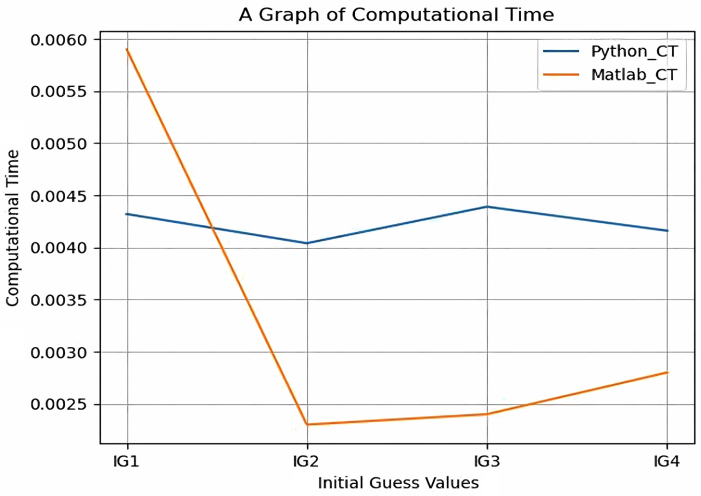


(a)

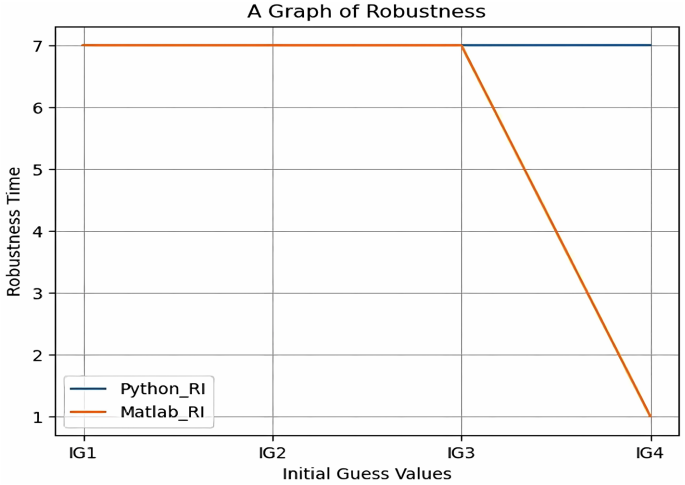


(b)

Figure 6. (a) CT against IG for Problem 1; (b) RI against IG for Problem 1.

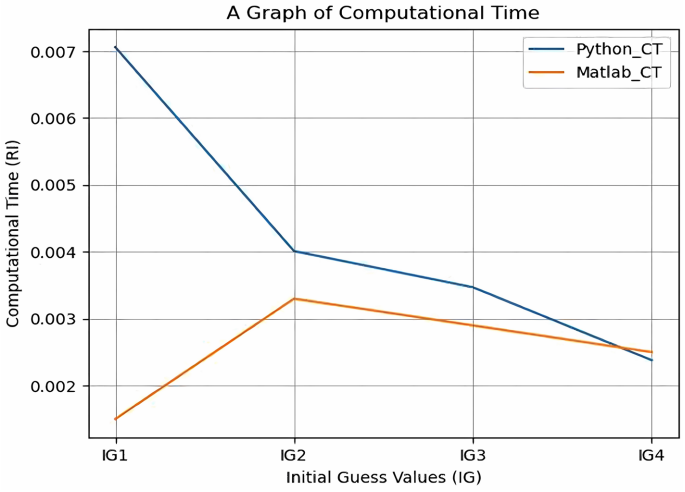


(a)

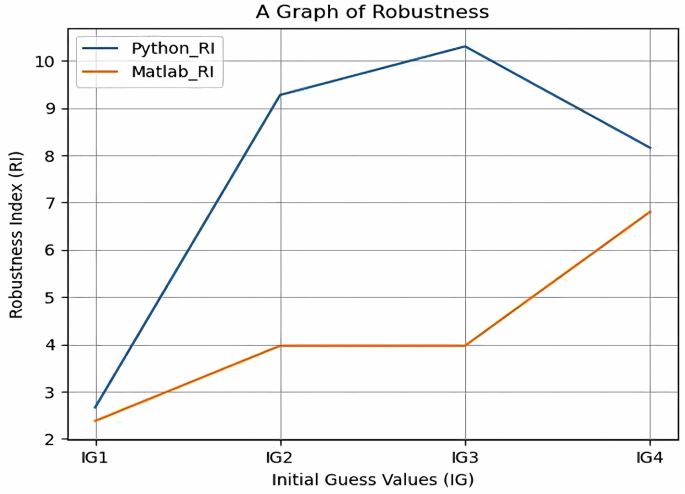


(b)

Figure 7. (a) CT against IG for Problem 2; (b) RI against IG for Problem 2.

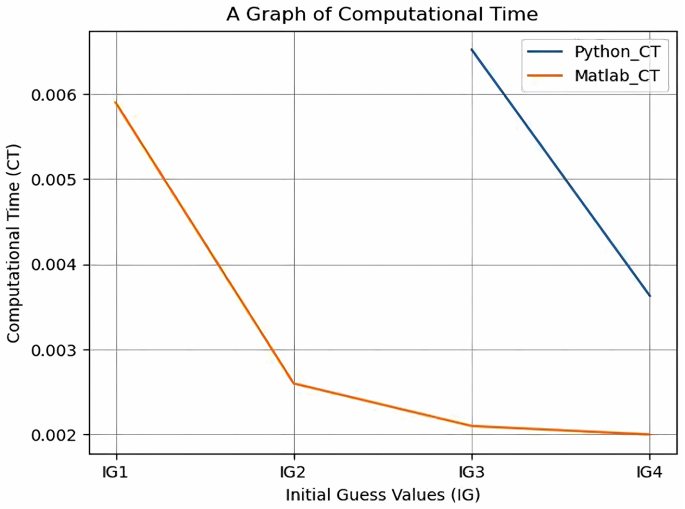


(a)

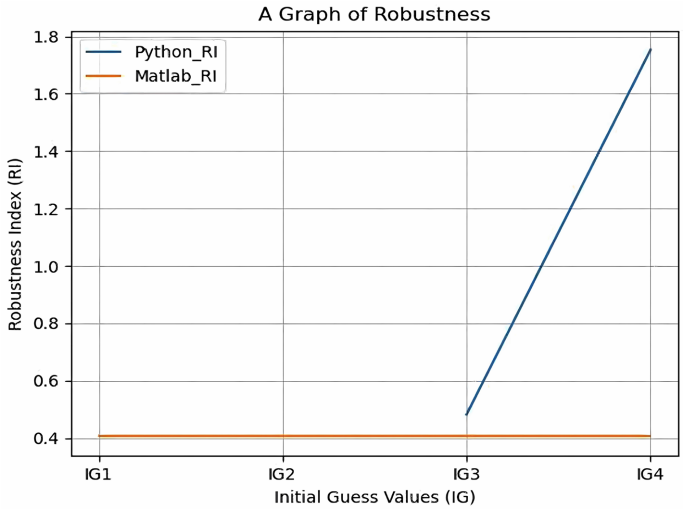


(b)

Figure 8. (a) CT against IG for Problem 3; (b) RI against IG for Problem 3.

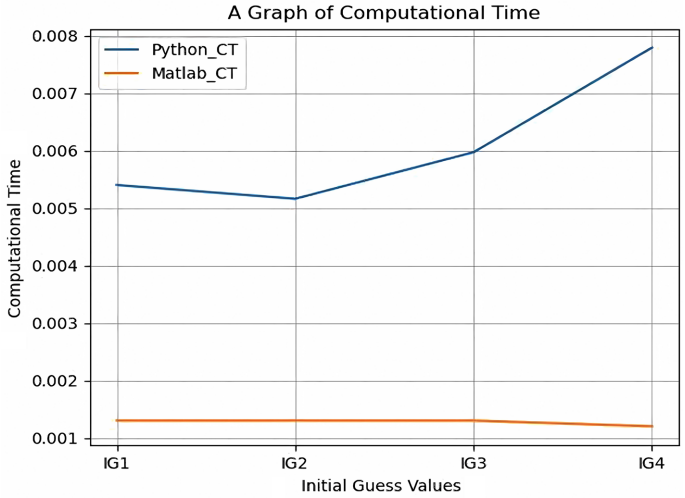


(a)

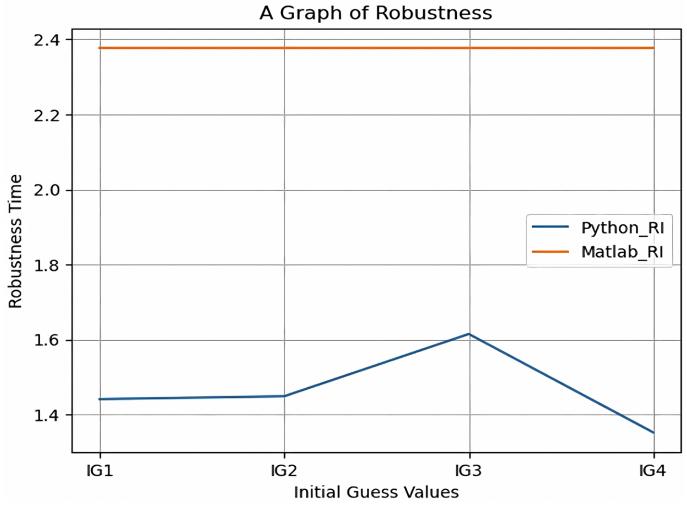


(b)

Figure 9. (a) CT against IG for Problem 4; (b) RI against IG for Problem 4.



(a)



(b)

Figure 10. (a) CT against IG for Problem 5; (b) RI against IG for Problem 5.

Table 5. Results of Dell Inspiron Laptop.

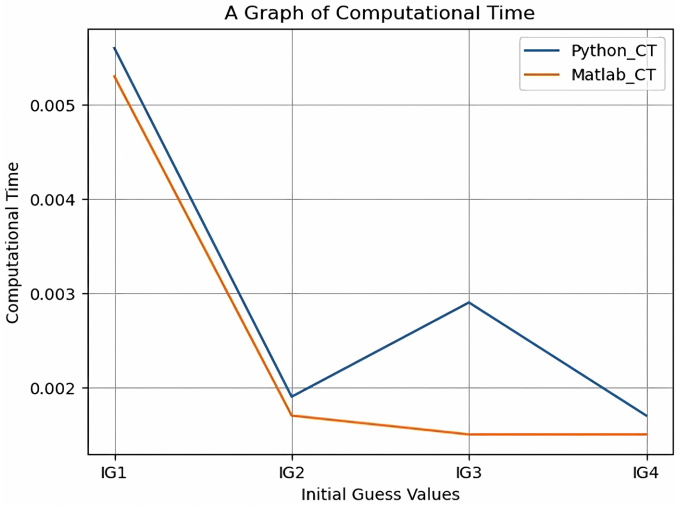
PROBLEM	PACKAGE	ROOT					ITERATIONS	CT	RI
		x_1	x_2	x_3	x_4	x_5			
	PYTHON								
	IG 1	0.9704	0.3364	1.7825	0.7077	−1.2506	78	0.0056	0.9711
	IG 2	0.9715	0.3372	1.7849	0.7073	−1.2508	83	0.0019	0.9711
	IG 3	0.9693	0.3366	1.7824	0.7149	−1.2519	115	0.0029	0.9711
	IG 4	0.8168	−0.3597	1.3220	0.1683	−0.7164	84	0.0017	1.2296
1	**MATLAB**								
	IG 1	0.9959	0.2987	1.7370	0.1282	−0.8380	66	0.0053	1.0835
	IG 2	0.9716	0.3363	1.7865	0.7119	−1.2533	83	0.0017	0.9710
	IG 3	0.9959	0.2987	1.7370	0.1282	−0.8380	67	0.0015	1.0835
	IG 4	0.9716	0.3363	1.7865	0.7119	−1.2533	76	0.0015	0.9710
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
	IG 1	0.7410	0.6166	−0.6872	1.4009	1.8392	82	0.0020	6.9986
	IG 2	0.7350	0.6294	−0.6719	1.4000	1.8439	76	0.0022	6.9979
	IG 3	0.7278	0.6153	−0.6710	1.4024	1.8434	62	0.0015	6.9980
	IG 4	0.7275	0.6153	−0.7004	1.4054	1.8441	67	0.0018	6.9996
2	**MATLAB**								
	IG 1	0.7326	0.6239	−0.6653	1.4011	1.8434	228	0.0028	6.9978
	IG 2	0.7326	0.6239	−0.6653	1.4011	1.8434	263	0.0029	6.9978
	IG 3	0.7326	0.6239	−0.6653	1.4011	1.8434	277	0.0030	6.9978
	IG 4	0.7326	0.6239	−0.6653	1.4011	1.8434	343	0.0038	6.9978
	SCILAB	-	-	-	-	-	-	-	-

Continued

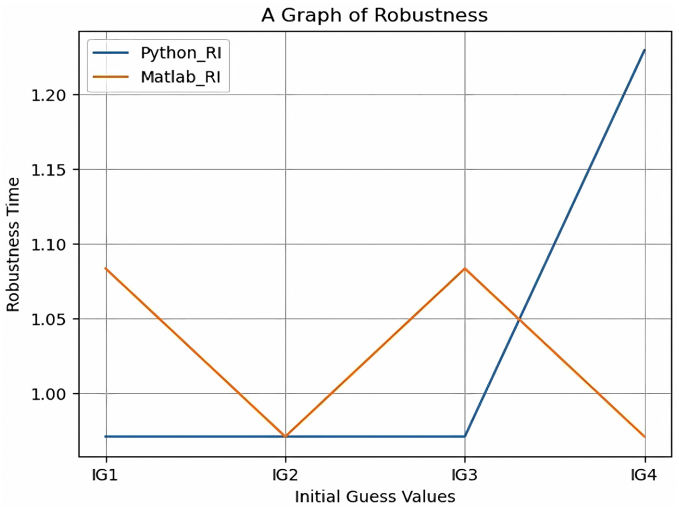
	PYTHON								
	IG 1	0.8870	0.5897	−0.4367	2.1678	1.8249	44	0.0011	8.1599
	IG 2	0.6935	−0.2732	0.2338	0.2515	0.4615	80	0.0014	2.3816
	IG 3	0.6658	−0.2763	0.2441	0.2322	0.4526	87	0.0017	2.3783
	IG 4	0.7762	−0.3105	0.0407	0.2461	0.5229	86	0.0015	2.4127
3	**MATLAB**								
	IG 1	0.6353	−0.2561	0.3129	0.2223	0.4475	72	0.0014	2.3772
	IG 2	−1.4358	0.2983	0.4158	1.1896	2.1513	400	0.0037	3.9702
	IG 3	−1.4359	0.2998	0.4148	1.1897	2.1519	400	0.0037	3.9702
	IG 4	0.6595	0.0822	−0.7859	0.9747	1.8351	400	0.0031	6.8038
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
	IG 1	0.99699	0.46632	−0.11986	1.10631	−0.37413	147	0.0011	7.76307
	IG 2	1.0810	0.54755	−0.30226	1.06365	−0.26264	161	0.0011	7.76307
	IG 3	1.05933	0.52320	−0.29394	1.07058	−0.27953	146	0.0011	7.76307
	IG 4	1.03781	0.88471	0.01949	0.78311	0.67484	77	0.0011	7.76307
4	**MATLAB**								
	IG 1	0.9518	0.4318	−0.0050	1.1308	−0.4562	179	0.0023	0.4052
	IG 2	0.9518	0.4318	−0.0050	1.1308	−0.4562	276	0.0029	0.4052
	IG 3	0.9518	0.4318	−0.0050	1.1308	−0.4562	225	0.0025	0.4052
	IG 4	0.9518	0.4318	−0.0050	1.1308	−0.4562	184	0.0023	0.4052
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
	IG 1	0.7932	−0.0009	1.1219	0.1665	0.0022	107	0.0028	1.4411
	IG 2	0.7946	−0.0031	1.1088	0.1789	0.0078	102	0.0022	1.4488
	IG 3	−0.2184	−0.4187	0.8618	0.8808	−1.0789	135	0.0027	1.682
	IG 4	1.8249	0.0236	1.8300	−0.0000	1.4933	162	0.0034	1.3522
5	**MATLAB**								
	IG 1	0.6353	−0.2561	0.3129	0.2223	0.4475	94	0.0018	2.3772
	IG 2	0.6353	−0.2561	0.3129	0.2223	0.4475	91	0.0015	2.3772
	IG 3	0.6353	−0.2561	0.3129	0.2223	0.4475	91	0.0014	2.3772
	IG 4	0.6353	−0.2561	0.3129	0.2223	0.4475	86	0.0014	2.3772
	SCILAB	-	-	-	-	-	-	-	-

its efficiency. Conversely, MATLAB excelled by achieving the lowest robustness index for problems 2 and 5, showcasing its capability to handle these particular challenges effectively (Figures 11(b)-15(b)).

Table 6 shows the results from the Dell Latitude Laptop.

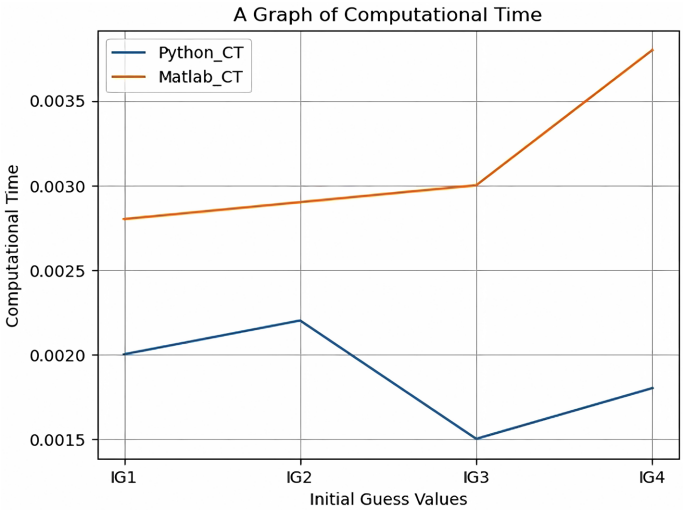


(a)

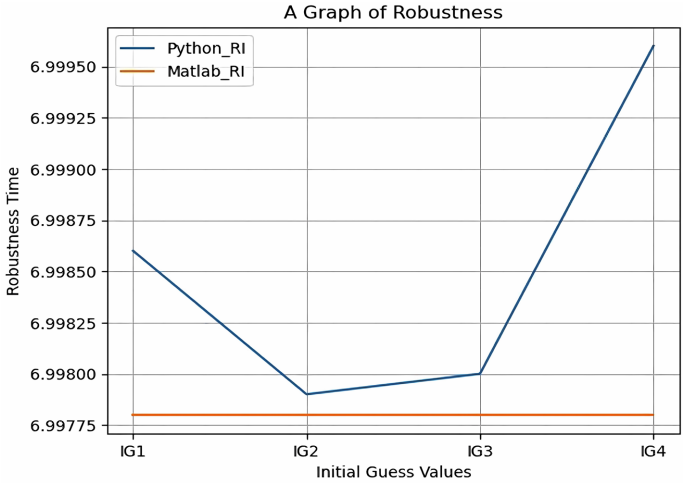


(b)

Figure 11. (a) CT against IG for Problem 1; (b) RI against IG for Problem 1.

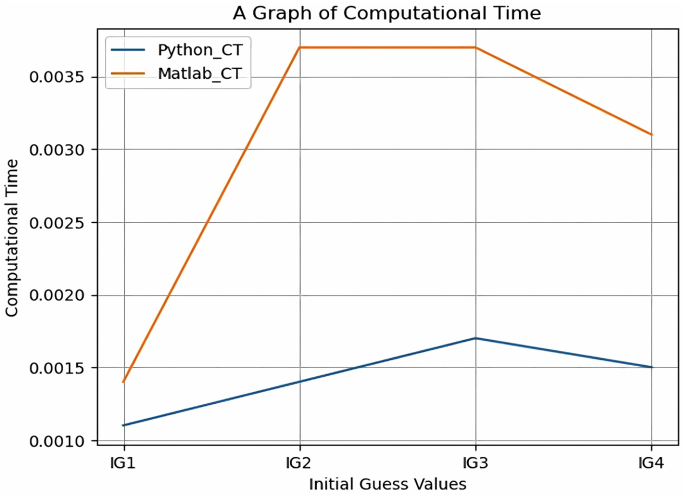


(a)

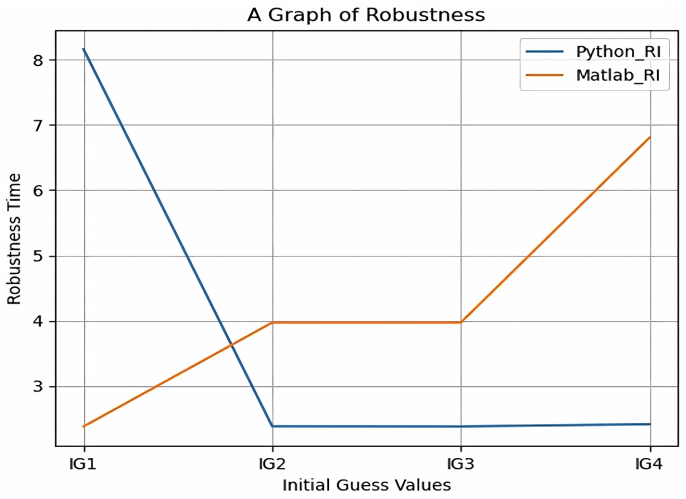


(b)

Figure 12. (a) CT against IG for Problem 2; (b) RI against IG for Problem 2.

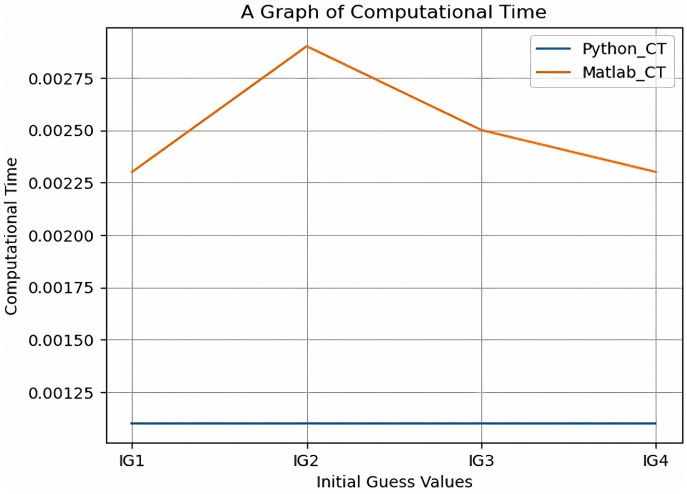


(a)

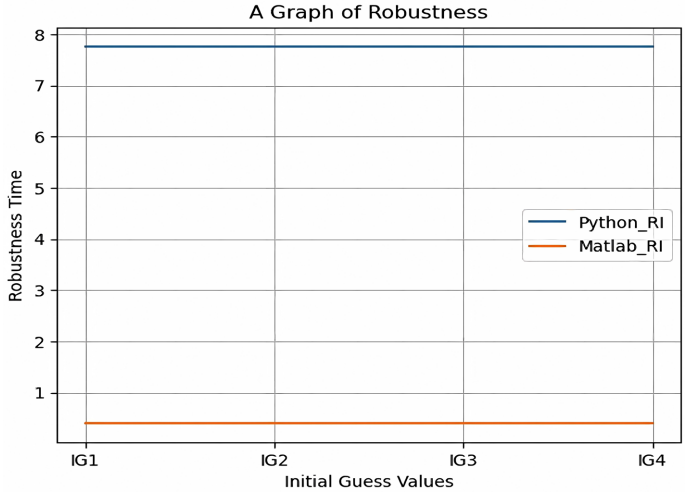


(b)

Figure 13. (a) CT against IG for Problem 3; (b) RI against IG for Problem 3.

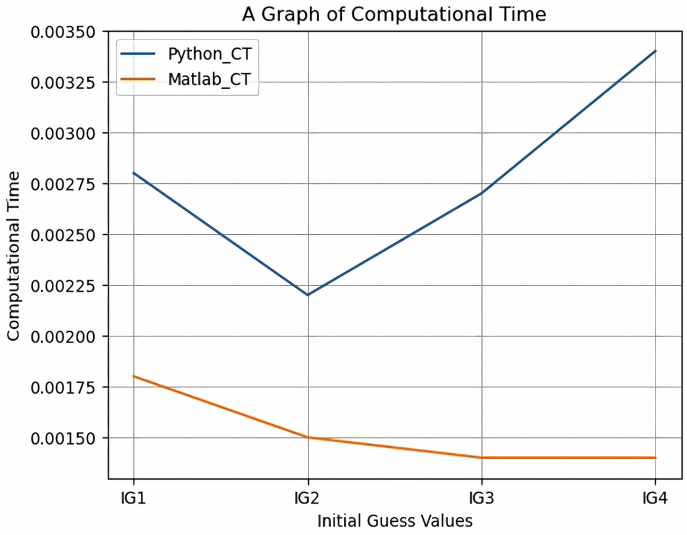


(a)

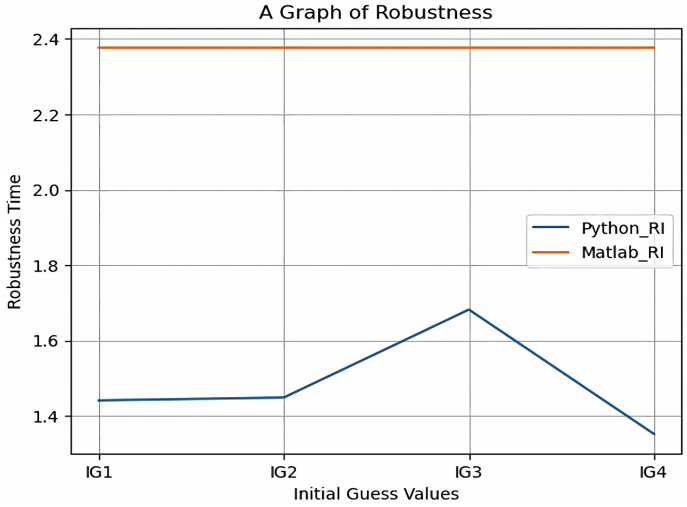


(b)

Figure 14. (a) CT against IG for Problem 4; (b) RI against IG for Problem 4.



(a)



(b)

Figure 15. (a) CT against IG for Problem 5; (b) RI against IG for Problem 5.

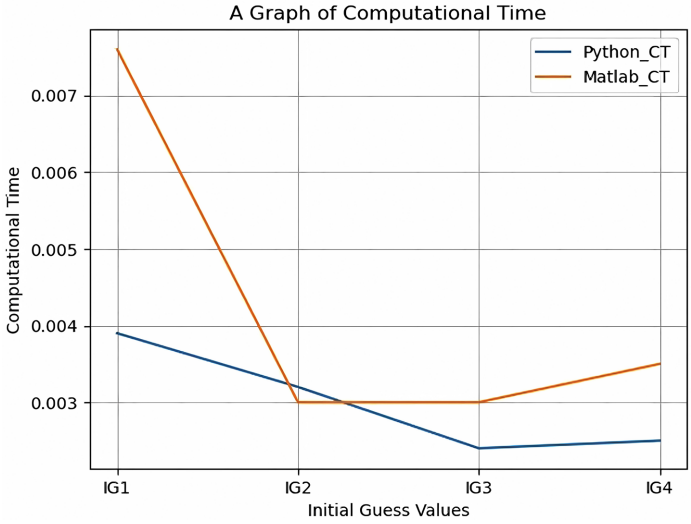
Table 6. Results of Dell Latitude Laptop.

PROBLEM	PACKAGE	ROOTS					ITERATIONS	CT	RI
		x_1	x_2	x_3	x_4	x_5			
	PYTHON								
	IG 1	0.9704	0.3364	1.7825	0.7077	−1.2506	78	0.0036	0.9711
	IG 2	0.9715	0.3372	1.7849	0.7073	−1.2508	83	0.0032	0.9711
	IG 3	0.9693	0.3366	1.7824	0.7149	−1.2519	115	0.0036	0.9711
	IG 4	0.8168	−0.3597	1.322	0.1683	−0.7164	84	0.0029	1.2296
1	**MATLAB**								
	IG 1	0.9959	0.2987	1.737	0.1282	−0.838	66	0.0013	1.0835
	IG 2	0.9716	0.3363	1.7865	0.7119	−1.2533	83	0.0015	0.971
	IG 3	0.9959	0.2987	1.737	0.1282	−0.838	67	0.0013	1.0835
	IG 4	0.9716	0.3363	1.7865	0.7119	−1.2533	76	0.0014	0.971
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
	IG 1	0.741	0.6166	−0.6872	1.4009	1.8392	82	0.0039	6.9986
	IG 2	0.735	0.6294	−0.6719	1.4	1.8439	76	0.0032	6.9979
	IG 3	0.7278	0.6153	−0.671	1.4024	1.8434	62	0.0024	6.998
	IG 4	0.7275	0.6153	−0.7004	1.4054	1.8441	67	0.0025	6.9996
2	**MATLAB**								
	IG 1	0.7326	0.6239	−0.6653	1.4011	1.8434	228	0.0076	6.9978
	IG 2	0.7326	0.6239	−0.6653	1.4011	1.8434	263	0.003	6.9978
	IG 3	0.7326	0.6239	−0.6653	1.4011	1.8434	277	0.003	6.9978
	IG 4	0.7326	0.6239	−0.6653	1.4011	1.8434	343	0.0035	6.9978
	SCILAB	-	-	-	-	-	-	-	-

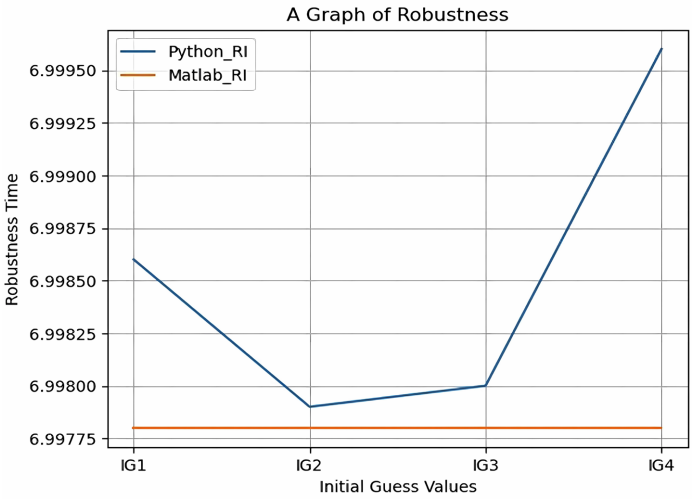
Continued

	PYTHON								
	IG 1	0.887	0.5897	−0.4367	2.1678	1.8249	44	0.0025	8.1599
	IG 2	0.6939	0.0791	−0.8103	0.9866	1.8349	64	0.0031	6.8047
	IG 3	−1.429	0.2733	0.4147	1.1896	2.1089	132	0.0044	3.976
	IG 4	0.6632	−0.2899	−0.4178	−0.3666	1.8521	64	0.0019	7.7631
3	**MATLAB**								
	IG 1	0.6353	−0.2561	0.3129	0.2223	0.4475	58	0.0016	2.3772
	IG 2	−1.4358	0.2983	0.4158	1.1896	2.1513	400	0.0039	3.9702
	IG 3	−1.4359	0.2998	0.4148	1.1897	2.1519	400	0.0035	3.9702
	IG 4	0.6595	0.0822	−0.7859	0.9747	1.8351	400	0.003	6.8038
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
	IG 1	0.99699	0.46632	−0.11986	1.10631	−0.37413	147	0.00191	7.76307
	IG 2	1.081	0.54755	−0.30226	1.06365	−0.26264	161	0.00191	7.76307
	IG 3	1.05933	0.5232	−0.29394	1.07058	−0.27953	146	0.00191	7.76307
	IG 4	1.03781	0.88471	0.01949	0.78311	0.67484	77	0.00191	7.76307
4	**MATLAB**								
	IG 1	0.9518	0.4318	−0.005	1.1308	−0.4562	196	0.0025	0.4052
	IG 2	0.9518	0.4318	−0.005	1.1308	−0.4562	276	0.003	0.4052
	IG 3	0.9518	0.4318	−0.005	1.1308	−0.4562	225	0.0025	0.4052
	IG 4	0.9518	0.4318	−0.005	1.1308	−0.4562	184	0.0024	0.4052
	SCILAB	-	-	-	-	-	-	-	-
	PYTHON								
	IG 1	0.7932	−0.0009	1.1219	0.1665	0.0022	107	0.0056	1.4411
	IG 2	0.7946	−0.0031	1.1088	0.1789	0.0078	102	0.004	1.4488
	IG 3	−0.2184	−0.4187	0.8618	0.8808	−1.0789	135	0.0042	1.682
	IG 4	1.8249	0.0236	1.83	0	1.4933	162	0.006	1.3522
5	**MATLAB**								
	IG 1	0.6353	−0.2561	0.3129	0.2223	0.4475	94	0.0017	2.3772
	IG 2	0.6353	−0.2561	0.3129	0.2223	0.4475	91	0.0015	2.3772
	IG 3	0.6353	−0.2561	0.3129	0.2223	0.4475	91	0.0015	2.3772
	IG 4	0.6353	−0.2561	0.3129	0.2223	0.4475	86	0.0014	2.3772
	SCILAB	-	-	-	-	-	-	-	-

Upon meticulous examination of the data garnered from the Dell Latitude laptop as shown in Table 6, an array of captivating trends about computational efficiency and robustness came to light. Python took the lead in computational time, offering expeditious solutions for problems 2, 3, and 4 as shown in Figures 16(a)-18(a). In contrast, MATLAB flexed its computational muscle, outperforming Python when it came to addressing problems 1 and 5 (Figure 19(b) and Figure 20(b)).

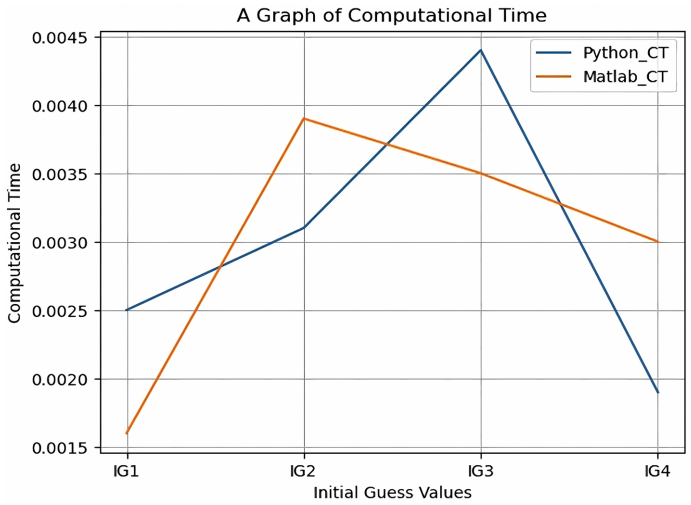


(a)

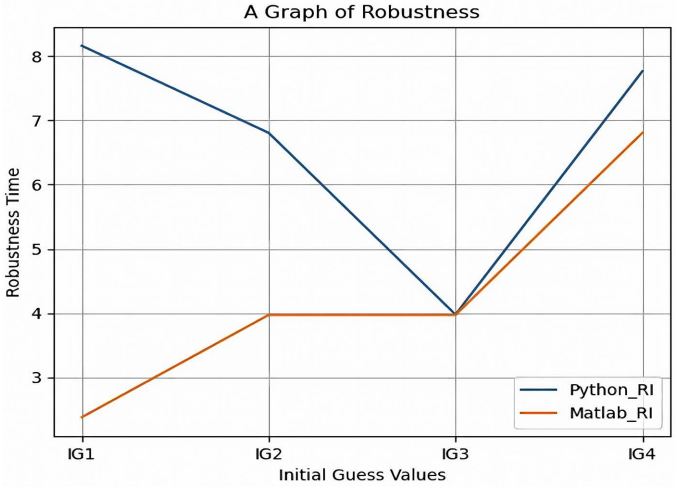


(b)

Figure 16. (a) CT against IG for Problem 2; (b) RI against IG for Problem 2.

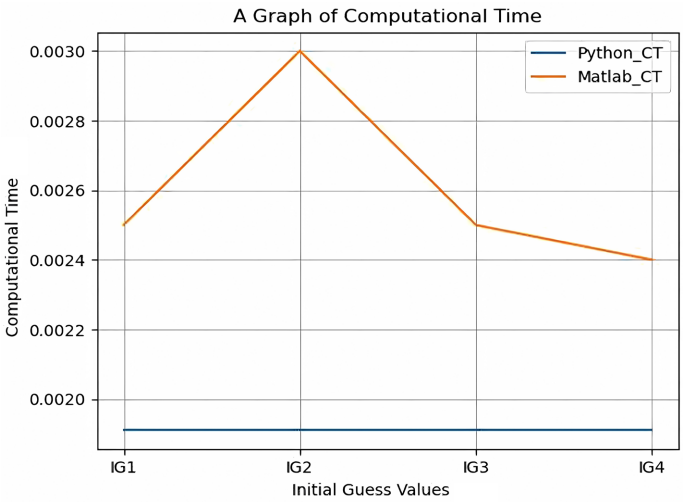


(a)

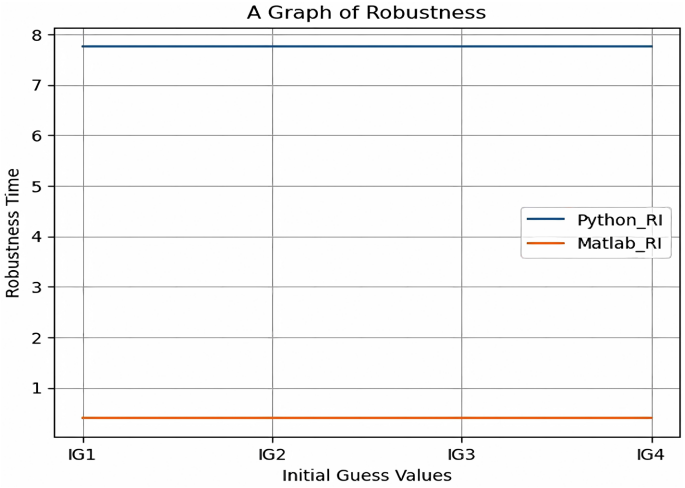


(b)

Figure 17. (a) CT against IG for Problem 3; (b) RI against IG for Problem 3.

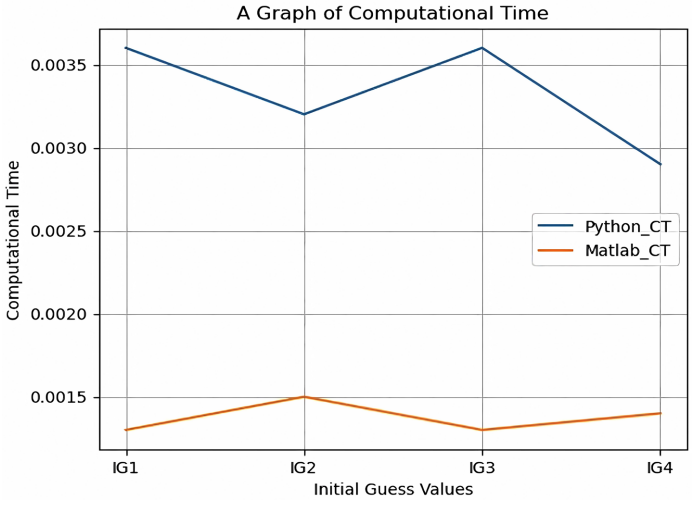


(a)

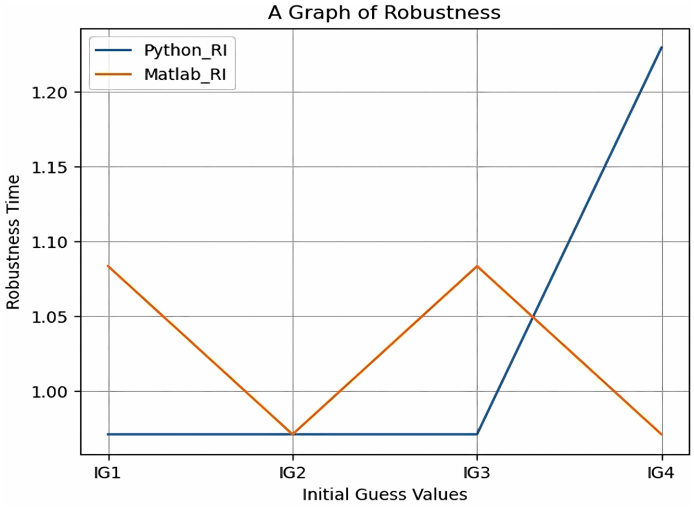


(b)

Figure 18. (a) CT against IG for Problem 4; (b) RI against IG for Problem 4.

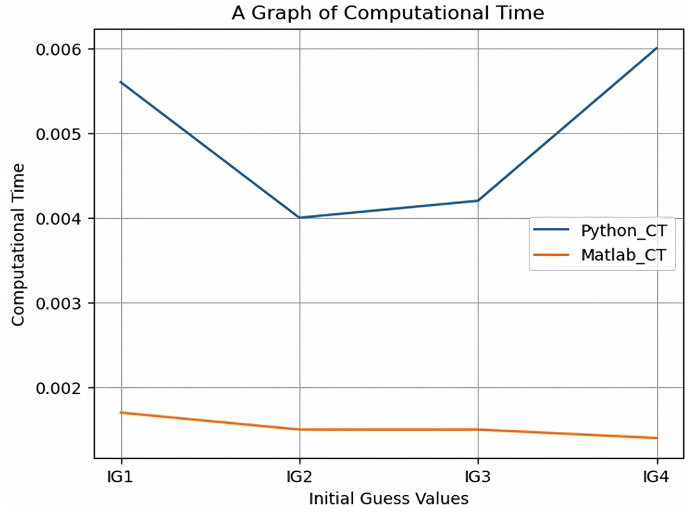


(a)

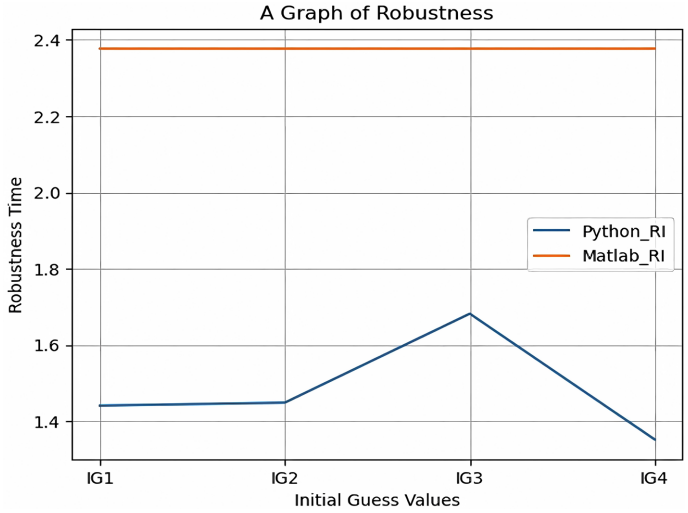


(b)

Figure 19. (a) CT against IG for Problem 1; (b) RI against IG for Problem 1.



(a)



(b)

Figure 20. (a) CT against IG for Problem 5; (b) RI against IG for Problem 5.

A similar narrative unfolded regarding the robustness index. Python exhibited a notably lower robustness index for problems 1 and 5 as in **Figure 19(a)** and **Figure 20(a)**, demonstrating its efficiency. Conversely, MATLAB shone by securing the lowest robustness index for problems 2, 3, and 4 (**Figures 16(b)-18(b)**), underscoring its adeptness in handling these specific challenges with finesse.

5. Conclusion and Recommendations

This study embarked on a comprehensive evaluation of three prominent mathematical software packages, namely Python, MATLAB, and Scilab, in their quest to solve nonlinear systems of equations with five unknown variables. Four primary objectives guided our research, which included comparing software performance using standardized benchmark problems, utilizing key performance metrics for quantitative evaluation, and assessing the impact of varying computer hardware specifications on software performance.

While numerous software packages are available for solving problems related to numerical analysis and computations, the most widely recognized and utilized ones include MATLAB, SCILAB, and Python. This prompted the authors of this research to conduct a comparative analysis of the results obtained from these software platforms.

Before the study began, it was expected that the computational time would vary due to the diverse computing environments anticipated to be used. However, this variation was not anticipated for the robustness index.

The utilization of the Broyden method as a numerical technique paved the way for meticulous comparisons. Across a diverse spectrum of computing environments represented by the HP Probook, HP Elitebook, Dell Inspiron, and Dell Latitude laptops, the performance of these software packages was assessed.

Firstly, our findings on the HP Probook laptop unveiled intriguing trends. Python consistently demonstrated shorter computational times, significantly out-

performing MATLAB. Python's computational time ranged from 0.0017950 seconds to 0.00465234 seconds, whereas MATLAB's spanned from 0.0448 seconds to 0.2690 seconds. In terms of the robustness index, Python showcased superiority by achieving a lower index in problems 3 and 5. Notably, for problem 1, Python mirrored MATLAB's index for initial guess values 1 and 2 but outperformed MATLAB for initial guess 3, while MATLAB displayed better performance for initial guess values 4.

The HP Elitebook laptop presented contrasting results. Here, MATLAB consistently exhibited notably shorter computational times than Python for all four benchmark problems, marking a significant divergence from the Probook HP device. In terms of robustness, MATLAB consistently yielded a lower index than Python across all benchmark problems except for problem 5. However, a distinct challenge emerged; Python failed to converge for problem 4 with initial guess values 2, whereas MATLAB successfully produced results for the same problem with those initial guess values.

The data collected from the Dell Inspiron laptop uncovered intriguing patterns. Python demonstrated superior computational efficiency for problems 1, 3, and 4, while MATLAB excelled in addressing problems 2 and 5. This pattern persisted in terms of the robustness index, where Python displayed a smaller index for problems 1, 3, and 5, while MATLAB secured the lowest index for problems 2 and 5.

The examination of data from the Dell Latitude laptop revealed intriguing insights into computational efficiency and robustness. Python demonstrated exceptional speed in solving problems 2, 3, and 4, making it a strong choice for these tasks. On the other hand, MATLAB showcased its computational prowess by outperforming Python in addressing problems 1 and 5. This trend was mirrored in the robustness index, with Python proving highly efficient for problems 1 and 5, while MATLAB excelled in ensuring robust solutions for problems 2, 3, and 4. These findings highlight the strengths of each tool in different problem-solving scenarios, emphasizing the importance of selecting the right programming language for specific computational needs.

In summation, this study has contributed valuable insights into the comparative capabilities of Python, MATLAB, and Scilab for tackling nonlinear systems of equations. Our findings underscore the significance of considering both software and hardware specifications in real-world applications. Depending on the specific problem and computational environment, the choice between Python and MATLAB can yield distinct advantages. We hope that these results will guide researchers and practitioners in selecting the most suitable tools for their unique challenges.

Conflicts of Interest

The authors declare no conflicts of interest that could influence the objectivity, integrity, or impartiality of the research presented in this paper.

References

[1] Azure, I. (2023) An Analysis of Solutions of Nonlinear Equations Using AI Inspired Mathematical Packages. *International Journal of Systems Science and Applied Mathematics*, **8**, 23-30. https://doi.org/10.11648/j.ijssam.20230802.12

[2] Downey, A.B. (2015) Think Python: How to Think like a Computer Scientist. Green Tea Press, St, Erie. http://greenteapress.com/thinkpython2/html/index.html

[3] Hahn, B. and Valentine, D.T. (2020) Essential MATLAB for Engineers and Scientists. Academic Press, Cambridge.

[4] Hanselman, D.C. and Littlefield, B.L. (2018) The Art of MATLAB. Cambridge University Press, Cambridge.

[5] Mahdy, A.M.S. (2022) A Numerical Method for Solving the Nonlinear Equations of Emden-Fowler Models. *Journal of Ocean Engineering and Science*. https://doi.org/10.1016/j.joes.2022.04.019

[6] Nagar, S. (2021) Introduction to Scilab. Notion Press, Chennai.

[7] Python Software Foundation (2021) Python 3.10.0 Documentation. https://docs.python.org/3/

[8] Rasheed, M., Shihab, S., Rashid, A., Rashid, T., Hamed, S.H.A. and Aldulaimi, M.A.H. (2021) An Iterative Method to Solve Nonlinear Equation. *Journal of Al-Qadisiyah for Computer Science and Mathematics*, **13**, 87. https://doi.org/10.29304/jqcm.2021.13.1.753

[9] Isaac, A., Stephen, T.B. and Seidu, B. (2021) A Comparison of Newly Developed Broyden-Like Methods for Solving System of Nonlinear Equations. *International Journal of Systems Science and Applied Mathematics*, **6**, 77-94. https://doi.org/10.11648/j.ijssam.20210603.11

[10] Rasheed, M., Rashid, A., Rashid, T., Hamed, S.H.A. and Al-Farttoosi, O.A.A. (2021) Application of Numerical Analysis for Solving Nonlinear Equation. *Journal of Al-Qadisiyah for Computer Science and Mathematics*, **13**, 70. https://doi.org/10.29304/jqcm.2021.13.1.752

[11] Biswa, N.D. (2012) Lecture Notes on Numerical Solution of Root-Finding Problems MATH 435.

[12] Martınez, J.M. (2000) Practical Quasi-Newton Methods for Solving Nonlinear Systems. *Journal of Computational and Applied Mathematics*, **124**, 97-121. https://doi.org/10.1016/S0377-0427(00)00434-9

[13] Mikac, M., Logožar, R. and Horvatić, M. (2022) Performance Comparison of Open Source and Commercial Computing Tools in Educational and Other Use—Scilab vs. MATLAB. *Tehnički Glasnik*, **16**, 509-518. https://doi.org/10.31803/tg-20220528171032

[14] Biegler, L.T. (2010) Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes. Society for Industrial and Applied Mathematics, Philadelphia. https://doi.org/10.1137/1.9780898719383

[15] Kelley, C.T. (1995) Iterative Methods for Linear and Nonlinear Equations. Society for Industrial and Applied Mathematics, Philadelphia. https://doi.org/10.1137/1.9781611970944

[16] Srivastava, R.B. and Srivastava, S. (2011) Comparison of Numerical Rate of Convergence of Bisection, Newton-Raphson's and Secant Methods. *Journal of Chemical, Biological and Physical Sciences* (*JCBPS*), **2**, 472.

[17] Xu, X.-B. (2022) An Algorithm on the Numerical Continuation of Asymmetric and

Symmetric Periodic Orbits Based on the Broyden Method and Its Application. *Chinese Astronomy and Astrophysics*, **63**, 401-421.

[18] Tolner, F., Barta, B. and Eigner, G. (2022) Comparison of Newton's and Broyden's Method as Nonlinear Solver in the Implementation of MFV-Robustified Linear Regression. 2022 *IEEE International Conference on Systems*, *Man*, *and Cybernetics* (*SMC*), Prague, 9-12 October 2022, 1518-1523. https://doi.org/10.1109/SMC53654.2022.9945222

[19] Ebelechukwu, O.C., Johnson, B.O., Michael, A.I. and Fidelis, A.T. (2018) Comparison of Some Iterative Methods of Solving Nonlinear Equations. *International Journal of Theoretical and Applied Mathematics*, **4**, 22-28. https://doi.org/10.11648/j.ijtam.20180402.11

[20] Ahmad, A.G. (2015) Comparative Study of Bisection and Newton-Raphson Methods of Root-Finding Problems. *International Journal of Mathematics Trends and Technology*, **19**, 121-129. https://doi.org/10.14445/22315373/IJMTT-V19P516

[21] Kazemi, M., Deep, A. and Nieto, J. (2023) An Existence Result with Numerical Solution of Nonlinear Fractional Integral Equations. *Mathematical Methods in the Applied Sciences*, **46**, 10384-10399. https://doi.org/10.1002/mma.9128