Advanced Newton Minimization

Shiva Surya Lolla

Date: December 11, 2023

Implementation details

Initializations

The code starts by initializing parameters that are tunable by the user to vary the performance of the implementation. These parameters include the least difference between the step updates (new x - current x), the maximum number of iterations of the step updates, the maximum number of sub-iterations of the line search method, the maximum number of sub-iterations of the trust region method, the amount added to the minimum eigenvalue to make a Hessian positive definite and the step size in the Armijo condition.

Function definition and initialization

This segment begins by establishing the analytical foundations required for function optimization. The function is first defined in terms of symbolic variables. The gradient and Hessian in terms of symbolic variables are then computed symbolically using MATLAB functions **gradient** and **hessian**.

An initial optimization variable guess is set and arrays to hold the series of variable values, function evaluations, and convergence metrics are set thus providing a comprehensive log of the optimization progress.

The execution of the optimization is then invoked via implementMultiNewtonMin, which is a custom function that embodies the multi-variable Newton's method enhanced with line search and trust region strategies. This function is provided with the initialization parameters from the previous sub-section and the function parameters from the current sub-section.

Main function

The implementMultiNewtonMin function encapsulates the core logic of the multi-variable Newton optimization process.

The iterative optimization begins with the initial guess and assesses convergence using the norm of the step difference as a metric. During each iteration, a subsidiary function singleStepMultiNewtonMin is called to perform a single optimization step, which could be a mix of Newton, line search, and trust region steps, based on the current point's properties.

Upon termination, whether by satisfying the convergence criterion or reaching the maximum number of iterations, the function plots the convergence of the variables' norms, the objective function values, and the logarithm of the error metrics. These visualizations help in understanding the optimization path and the algorithm's performance.

Finally, the function concludes by printing a summary table detailing the types of steps taken and their iteration counts, providing a concise statistical overview of the optimization journey. The approximate root, or solution, is also displayed, which is the desired result of the process.

Single step minimization function

The singleStepMultiNewtonMin function is designed to compute a single iteration within the broader multi-variable Newton optimization framework. This function decides the optimal step type—Newton, line search, or trust region—based on the current state of the optimization.

At the start of each iteration, the function initializes flags and counters to track the progress and decision-making throughout the function. It constructs a structured representation of the current variable values and then calculates the current function value, gradient, and Hessian using symbolic substitution.

The function then checks the positive definiteness of the Hessian matrix, as a Newton step is only taken in the direction of a positive definite Hessian. If the Hessian is not positive definite, the function adjusts it by adding a multiple of the identity matrix, with the multiple being determined by the minimum eigenvalue of the Hessian and a user-defined small positive value for numerical stability while Hessian inversion (this has been chosen to be 1e-4 in this code).

The backtracking line search function is called with a maximum number of sub-iterations. These sub-iterations ensure that the implementation of line search exists if there is a tendency for the optimization to get stuck. The outcome of this process determines the type of step taken; a single sub-iteration suggests a Newton step, while multiple sub-iterations indicate a line search step. Note that the Newton step is implemented as a part of the backtracking line search function. This has been done for the ease of writing code because the Newton step is the very first step of backtracking line search which is taken if the Armijo condition is satisfied in the very first sub-iteration.

If the maximum number of line search iterations is reached, the function escalates the process to a trust region update. This update is more sophisticated, as it constrains the step within a certain radius, and uses the dogleg path strategy to navigate through challenging regions of the function's landscape.

The function flags an exit if the trust region method also reaches its maximum number of iterations, which means a potential issue with convergence at the current point and indicates a failure to find a solution for this case. Upon completing the step, the function returns the new point, an exit flag indicating if a maximum iteration limit was hit, the function value at the new point, the number of iterations for line search and trust region methods, and the type of step taken.

Line Search and Polynomial Fitting Functions

This section of the code introduces three crucial functions `quadraticFitting`, `cubicFitting`, and `BacktrackingSearch`—each essential in the line search process of the multi-variable Newton optimization method.

Quadratic Fitting

The `quadraticFitting` function is utilized for either line search or trust region lambda updates. It fits a quadratic polynomial based on the function's current value, its derivative at this point, and the function value at a potential next point. This step is needed to estimate an appropriate step size (λ), to ensure a reduction in the function value.

Cubic Fitting

`cubicFitting` extends the fitting to a cubic polynomial, especially useful in line search lambda updates for enhanced accuracy. It is called when the quadratic fitting has not given a satisfactory estimate of the step size (λ). It incorporates the previous two λ values and their respective function evaluations, calculating the coefficients of a cubic polynomial. These coefficients are then used to determine a new λ value.

Backtracking Line Search

The `BacktrackingSearch` function implements the backtracking line search algorithm, employing the Armijo condition. It adjusts λ to locate a point where the function value decreases sufficiently. Beginning with a full Newton step, it employs quadratic or cubic fitting for further step size refinement when necessary. This is particularly crucial in scenarios where a full Newton step might not be optimal.

The λ updates are constrained by bounds to avoid excessive or insufficient step sizes. Upon completion, the function computes the updated point and returns it along with the count of iterations.

Trust Region Update with Dogleg Method

The `TrustRegionDoglegUpdate` function is responsible for performing the trust region update using the double dogleg method.

Initialization

This function commences by printing the optimization's entry into the trust region phase, signifying a shift from other optimization strategies. It starts by determining the Newton direction, which is the primary direction of the step if no constraints were imposed.

Trust Region Radius

The function then calculates the initial trust region radius which defines the maximum allowable step size in the current iteration. It is initialized as the length of the Cauchy step (Powell 1970a from the book), which provides a first estimate of the step length in the steepest descent direction.

Path Strategy

The dogleg path, which is a piecewise linear path combining the steepest descent direction and the Newton direction is then computed by combining the Cauchy direction and scaled dogleg direction. This path is used to find an approximate solution within the trust region that sufficiently reduces the function value.

Iteration and Alpha Condition

Within each iteration, the function checks if the chosen step satisfies the alpha condition. If the condition is not met, the function performs a quadratic fit to backtrack and refine the trust region radius, thereby adjusting the step size.

Termination and Output

The iteration continues until either an acceptable step is found based on the alpha condition or the maximum number of sub-iterations for the trust region update is reached. If the maximum iteration count is exceeded, the function sets an exit flag indicating a potential convergence issue. Upon successful completion, the function computes and returns the new point and the number of sub-iterations taken, to understand the trust region step's effectiveness.

Results

The objective of this study is to implement and assess the quasi-Newton method for unconstrained minimization. The core algorithm combines the robustness of Newton's method with a backtracking line search and trust region strategy, dynamically switching between local and global step strategies.

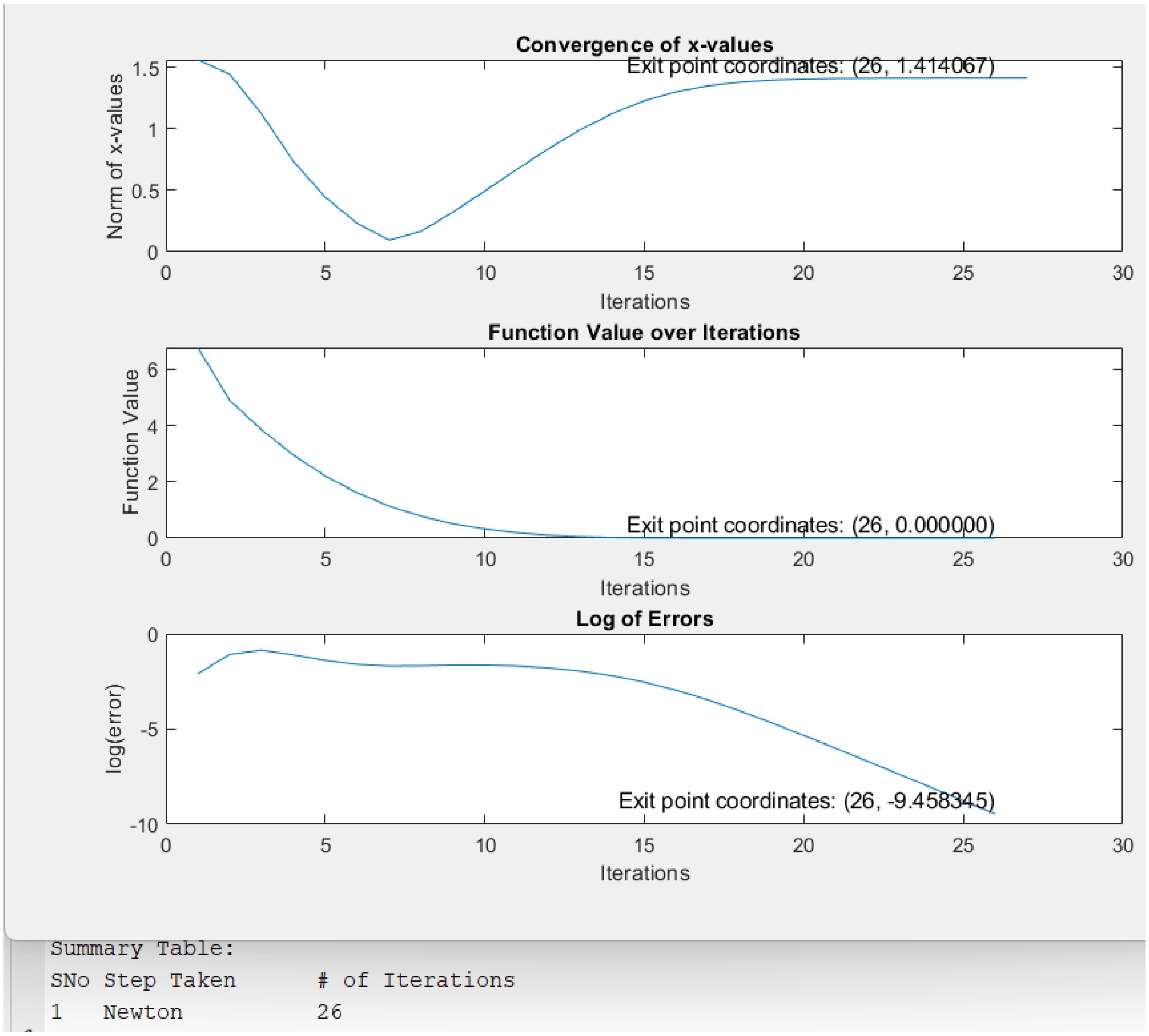
The developed MATLAB program has been tested on three benchmark functions: the Extended Rosenbrock Function with $n = 2$, the Extended Powell Singular Function with $n = 4$, and the Wood Function. The analysis focuses on the convergence properties and the efficacy of the hybrid approach, considering different initial guesses for each function.

- **Extended Rosenbrock Function** ($n = 2$): Defined as $f_{\text{rosenbrock}}(x_1, x_2) = 10(x_2 - x_1^2)^2 + (1 - x_1)^2$.
- **Extended Powell Singular Function** ($n = 4$): Defined as $f_{\text{powell}}(x_1, x_2, x_3, x_4) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$.
- **Wood Function**: Defined as $f_{\text{wood}}(x_1, x_2, x_3, x_4) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$.

The performance indicators included the approximate solution accuracy and the observed rate of convergence. The goal was to evaluate how the initial guess proximity to the expected solution influenced the algorithm's performance.

2a

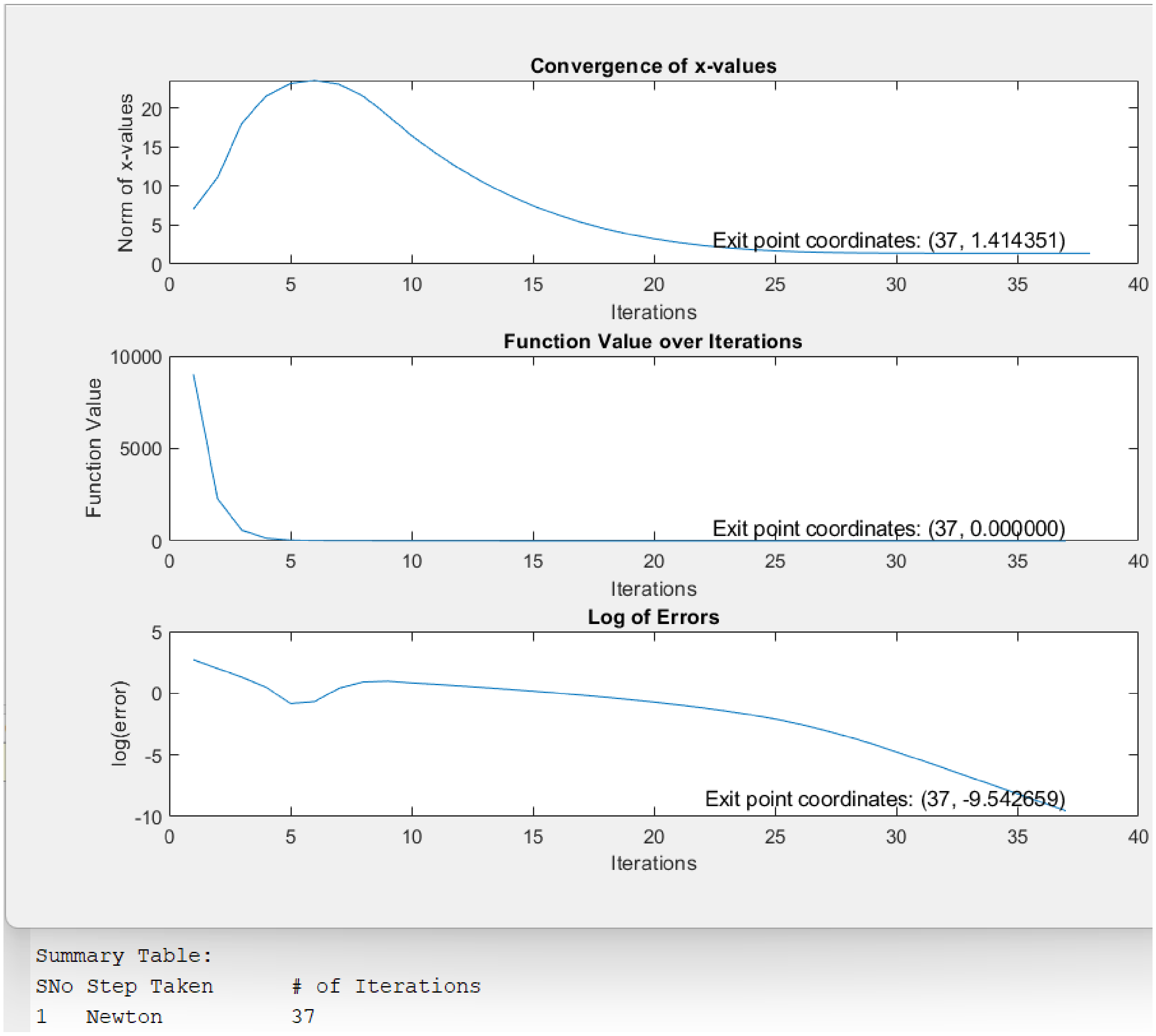
For Extended Rosenbrock Function with n = 2 we have the following results
For the initial condition $x0_{rosenbrock} = [-1.2; 1]$, The root is approximately:
0.99996697593
0.99992926169


Convergence of x-values
Exit point coordinates: (26, 1.414067)
Norm of x-values
Iterations
Function Value over Iterations
Function Value
Exit point coordinates: (26, 0.000000)
Iterations
Log of Errors
log(error)
Exit point coordinates: (26, -9.458345)
Iterations

```
Summary Table:
SNo Step Taken      # of Iterations
1   Newton          26
```

Figure 1

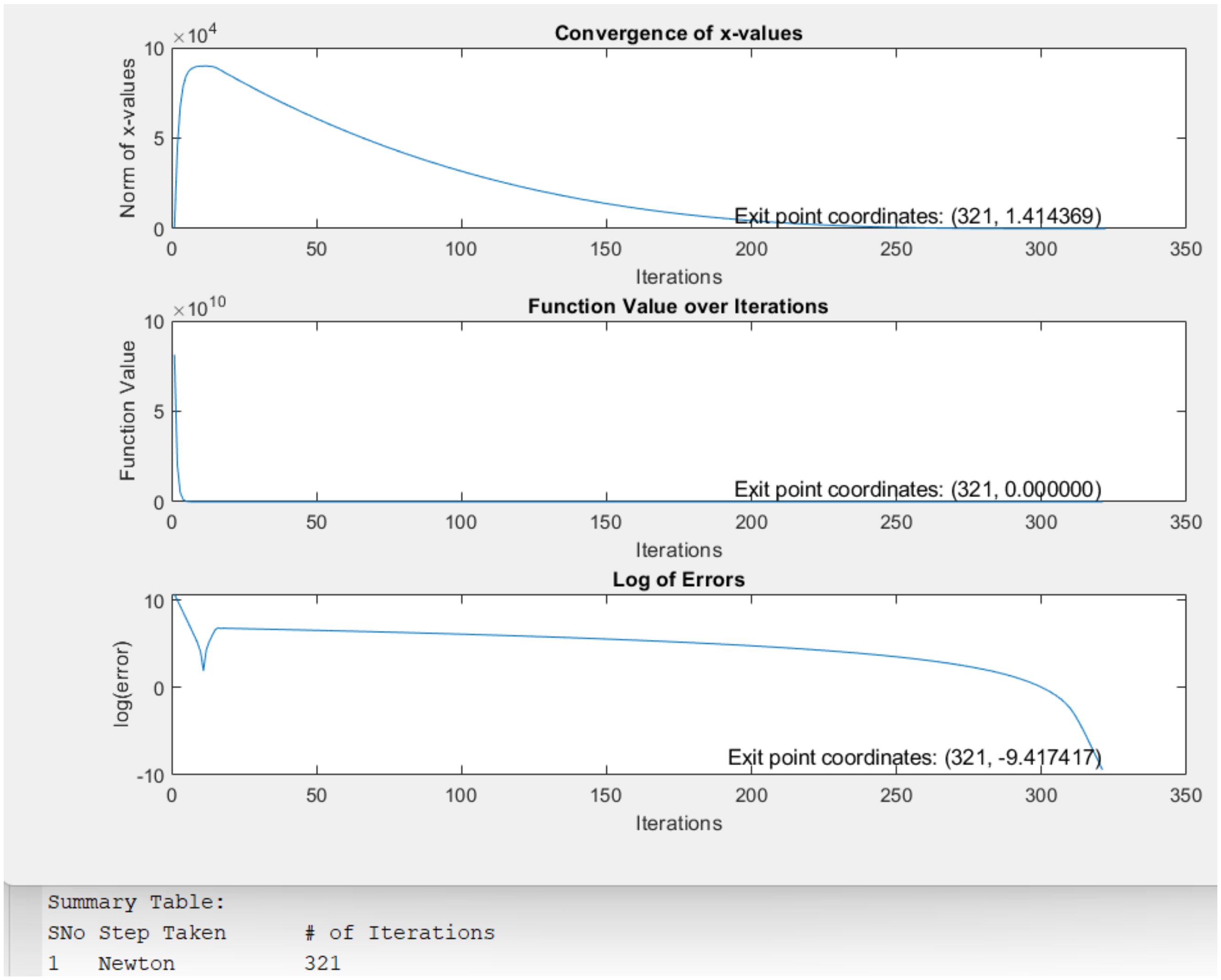
For the initial condition $x0_{rosenbrock} = [5; -5]$, The root is approximately:
1.00003400537
1.0000631812



```
Summary Table:
SNo Step Taken      # of Iterations
1   Newton          37
```

Figure 2

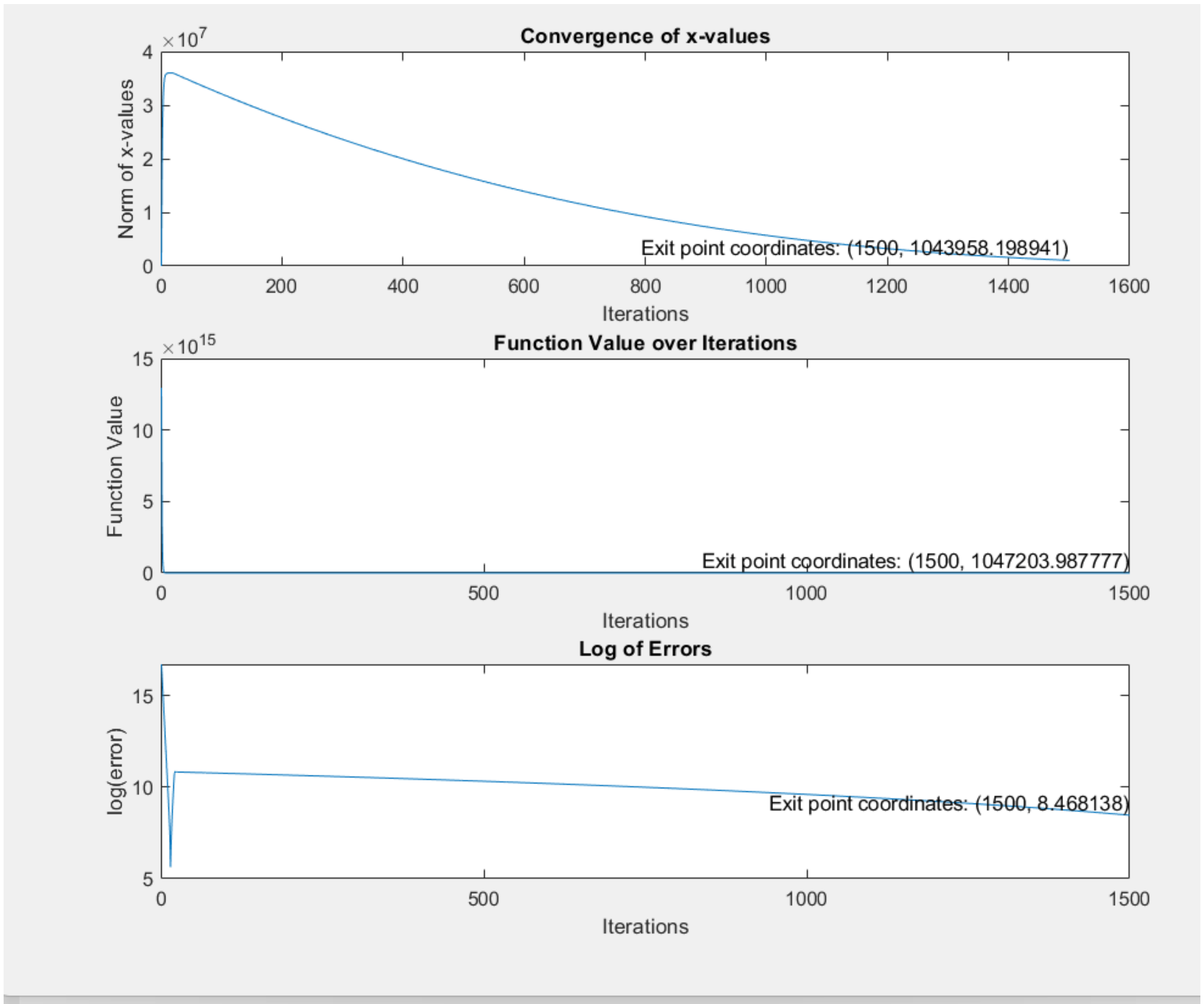
For the initial condition $x0_{rosenbrock} = [300; -150]$, The root is approximately:
1.00003854416
1.00007161441



```
Summary Table:
SNo Step Taken       # of Iterations
1   Newton           321
```

Figure 3

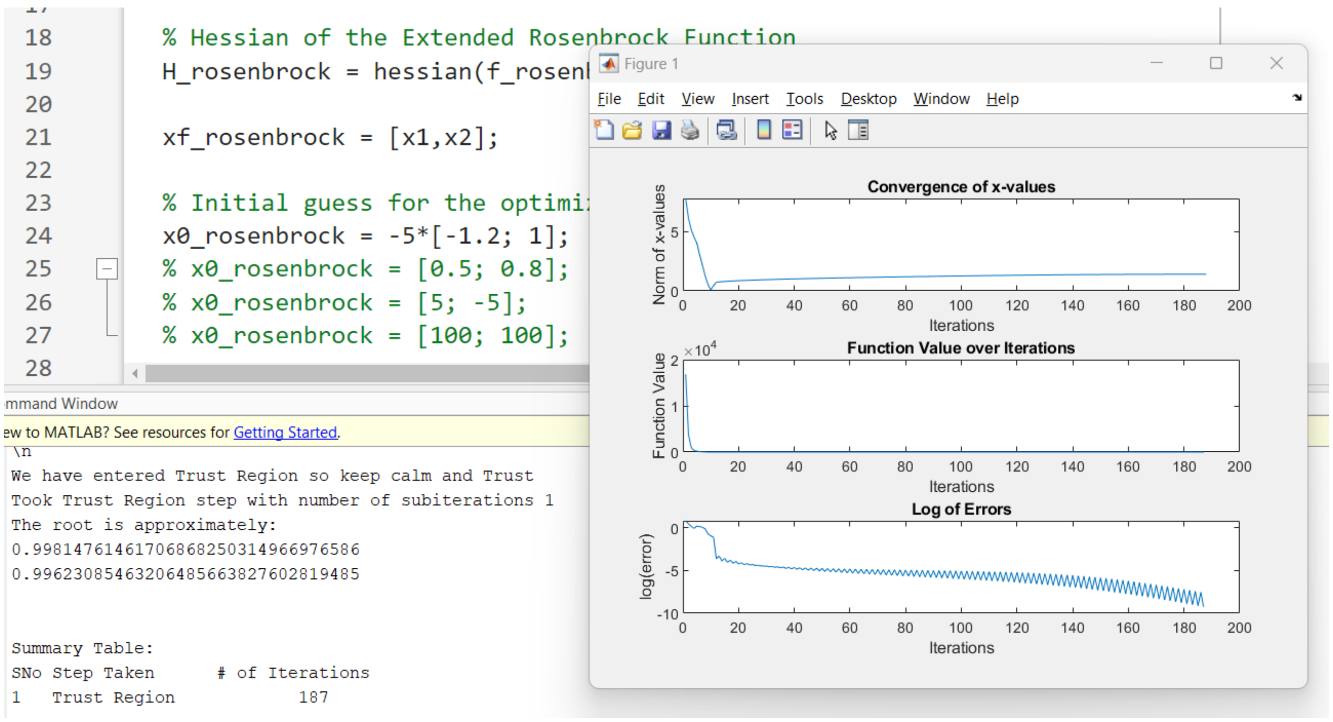
For the initial condition $x0_{rosenbrock} = 5000 * [-1.2; 1]$, The root is approximately:
-1019.415496334
1039197.0580497



```
Summary Table:
SNo Step Taken       # of Iterations
1   Newton           1500
```

Figure 4

The rate of convergence for the Extended Rosenbrock Function appears to be quadratic in all cases. As per my observation, the Rosenbrock function is not going into the Line Search or Trust Region updates and is continuing with the Newton step no matter what the initial conditions are. I experimented with the alpha parameter to possibly redirect it towards line search but even that did not help.



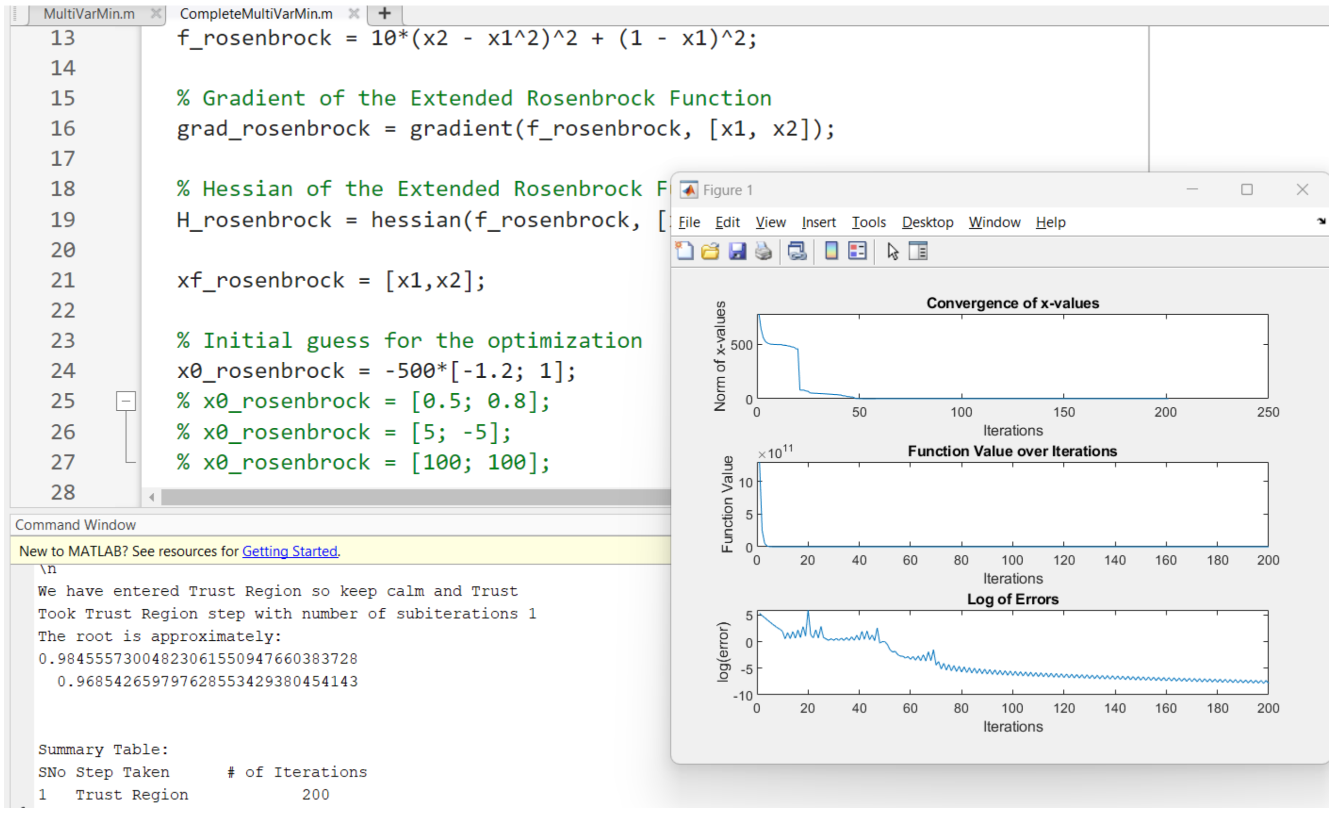
```
% Hessian of the Extended Rosenbrock Function
H_rosenbrock = hessian(f_rosenl

xf_rosenbrock = [x1,x2];

% Initial guess for the optimi
x0_rosenbrock = -5*[-1.2; 1];
% x0_rosenbrock = [0.5; 0.8];
% x0_rosenbrock = [5; -5];
% x0_rosenbrock = [100; 100];
```

mmand Window

ew to MATLAB? See resources for Getting Started.

```
\n
We have entered Trust Region so keep calm and Trust
Took Trust Region step with number of subiterations 1
The root is approximately:
0.99814761461706868250314966976586
0.99623085463206485663827602819485


Summary Table:
SNo Step Taken       # of Iterations
1    Trust Region            187
```

Figure 5



MultiVarMin.m CompleteMultiVarMin.m

```
f_rosenbrock = 10*(x2 - x1^2)^2 + (1 - x1)^2;

% Gradient of the Extended Rosenbrock Function
grad_rosenbrock = gradient(f_rosenbrock, [x1, x2]);

% Hessian of the Extended Rosenbrock F
H_rosenbrock = hessian(f_rosenbrock, [

xf_rosenbrock = [x1,x2];

% Initial guess for the optimization
x0_rosenbrock = -500*[-1.2; 1];
% x0_rosenbrock = [0.5; 0.8];
% x0_rosenbrock = [5; -5];
% x0_rosenbrock = [100; 100];
```

Command Window

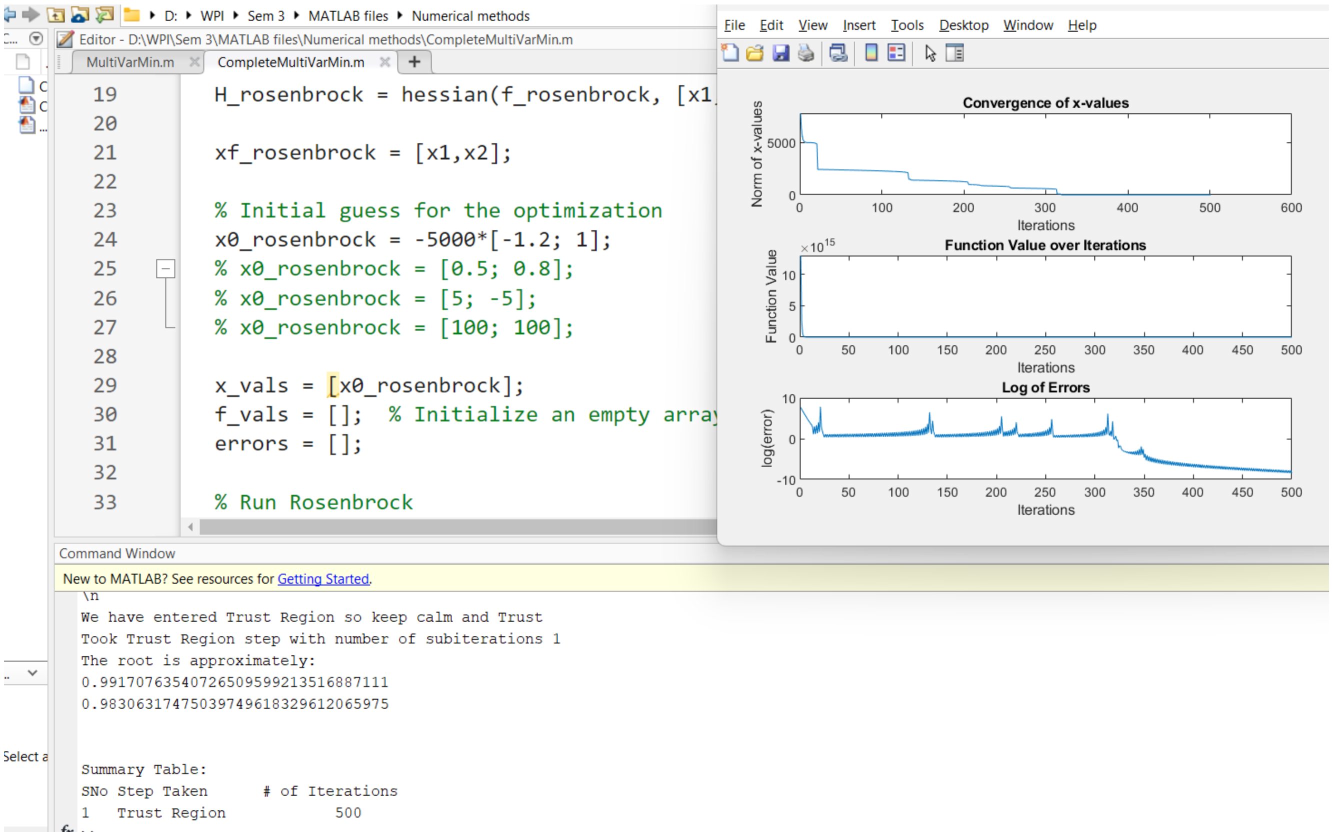
New to MATLAB? See resources for Getting Started.

```
\n
We have entered Trust Region so keep calm and Trust
Took Trust Region step with number of subiterations 1
The root is approximately:
0.98455573004823061550947660383728
  0.96854265979762855342938045414143


Summary Table:
SNo Step Taken       # of Iterations
1    Trust Region            200
```

Figure 6

Therefore, I implemented just the trust region approach (disabling Newton and Line Search) for the Rosenbrock function. This made it enter the trust region and the trust region method solves it accurately.

When the initial point is very far from the solution (Figure 4), no convergence is observed and the code exits and plots the output after the maximum number of iterations. This indicates that really bad initializations are not working well for the Rosenbrock function. Surprisingly, the trust region approach solved the issue in case of Figure 4 elegantly.


D: ▸ WPI ▸ Sem 3 ▸ MATLAB files ▸ Numerical methods
Editor - D:\WPI\Sem 3\MATLAB files\Numerical methods\CompleteMultiVarMin.m
MultiVarMin.m
CompleteMultiVarMin.m
File Edit View Insert Tools Desktop Window Help
Convergence of x-values
Norm of x-values
Iterations
Function Value over Iterations
Function Value
×10^15
Log of Errors
log(error)
Command Window
New to MATLAB? See resources for Getting Started.

```
H_rosenbrock = hessian(f_rosenbrock, [x1

xf_rosenbrock = [x1,x2];

% Initial guess for the optimization
x0_rosenbrock = -5000*[-1.2; 1];
% x0_rosenbrock = [0.5; 0.8];
% x0_rosenbrock = [5; -5];
% x0_rosenbrock = [100; 100];

x_vals = [x0_rosenbrock];
f_vals = [];  % Initialize an empty array
errors = [];

% Run Rosenbrock
```

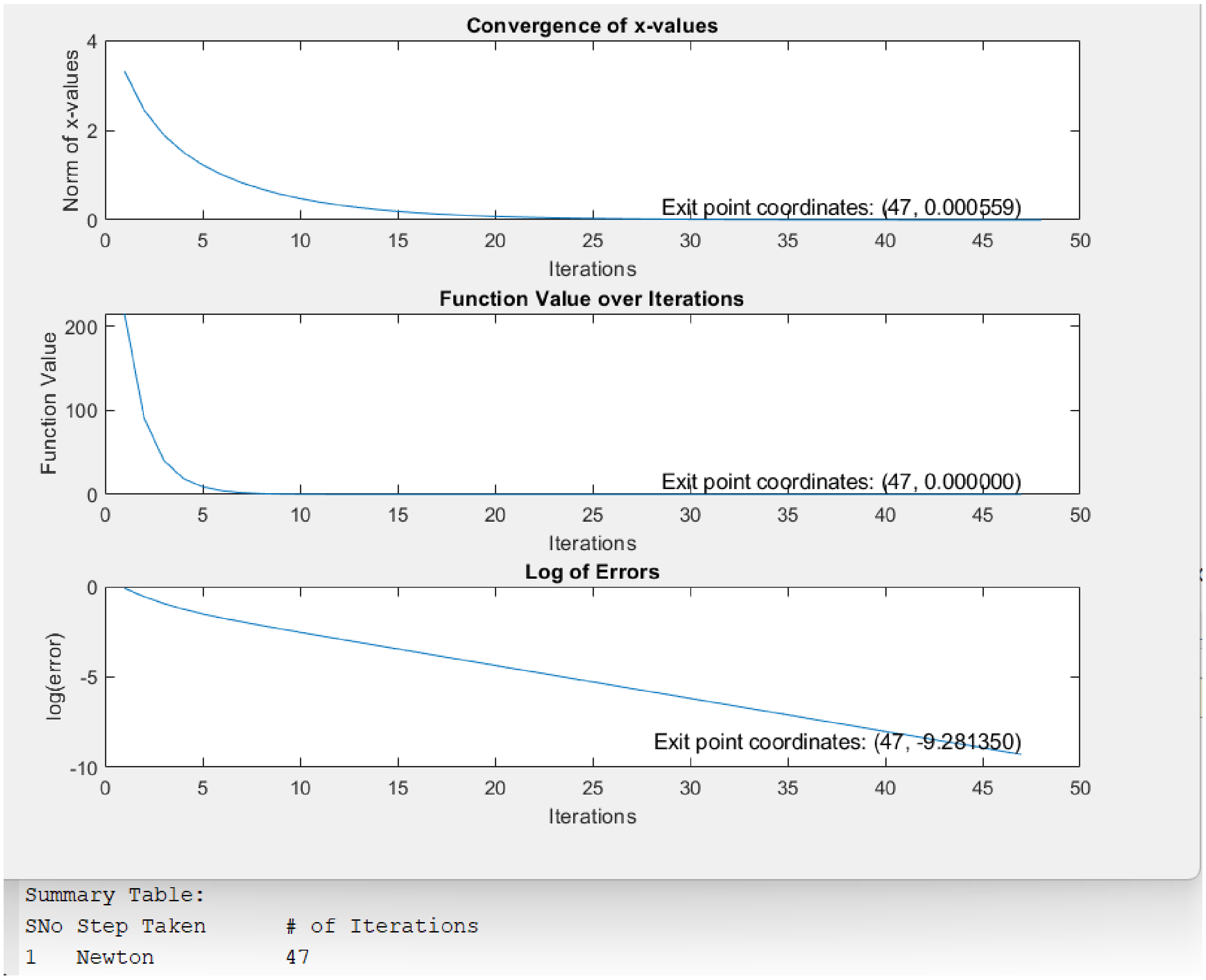
```
\n
We have entered Trust Region so keep calm and Trust
Took Trust Region step with number of subiterations 1
The root is approximately:
0.99170763540726509599213516887111
0.98306317475039749618329612065975


Summary Table:
SNo Step Taken      # of Iterations
1   Trust Region            500
```

Figure 7

Also, in general for the Rosenbrock function, initializations far from the solution took more number of iterations to converge.

2b

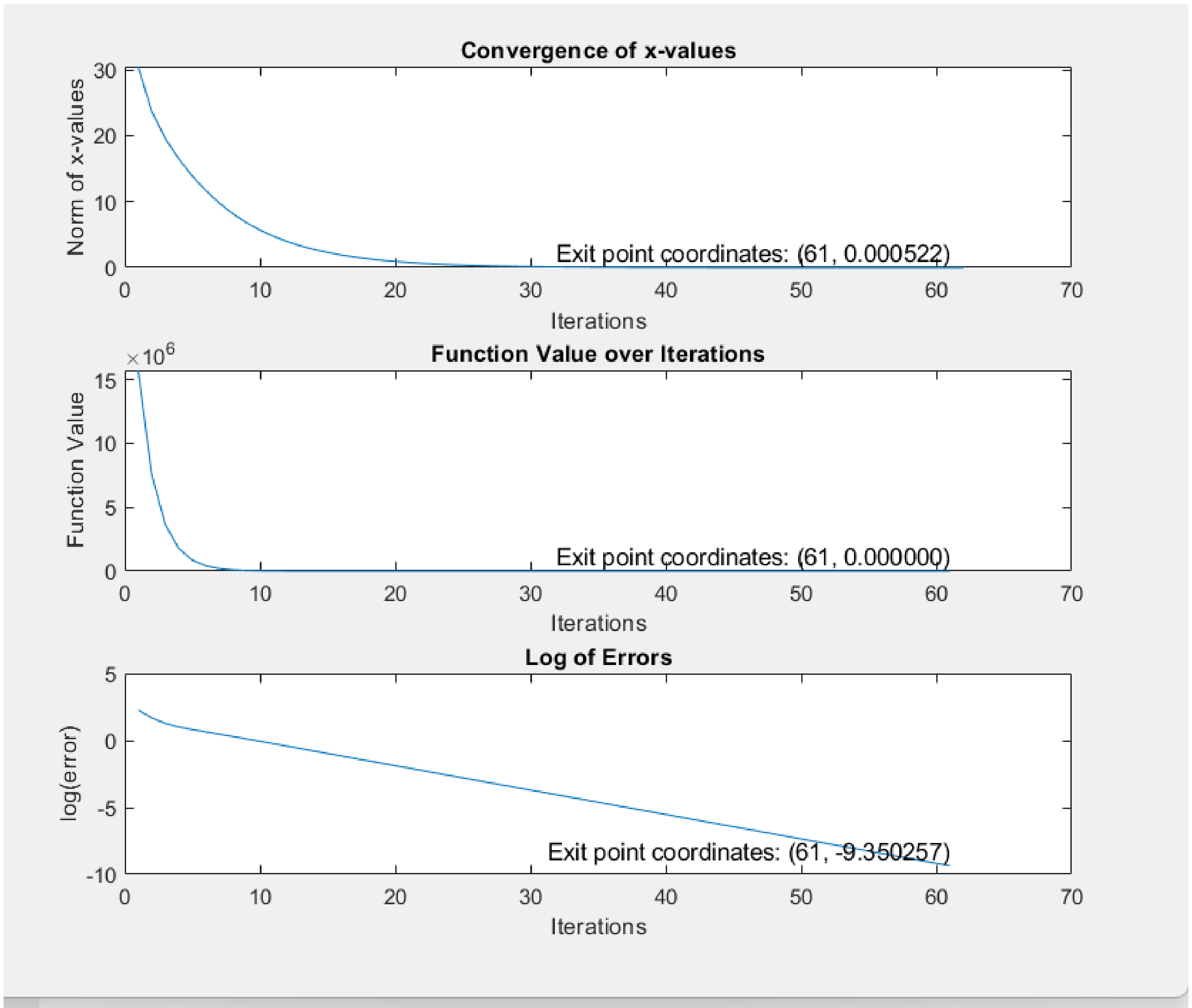
For Extended Powell Singular Function with n = 4 function, we have the following results
For the initial condition $x0_{powell} = [3; -1; 0; 1]$, The root is approximately:
0.000452097541
-0.000045209754
0.0000723356066
0.0000723356066

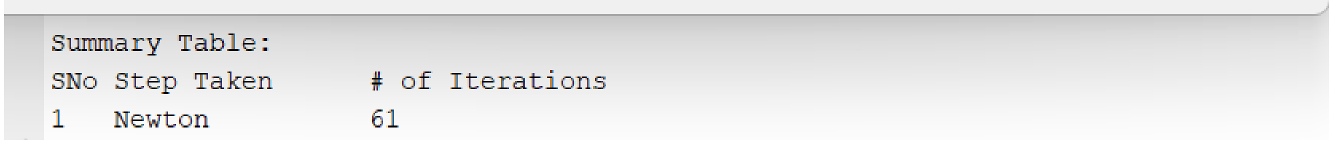


```
Summary Table:
SNo Step Taken       # of Iterations
1    Newton          47
```

Figure 8

For the initial condition $x0_{powell} = [10; 11; -9; -25]$, The root is approximately:
0.000288741064
-0.0000288741064
-0.000228880112
-0.000228880112

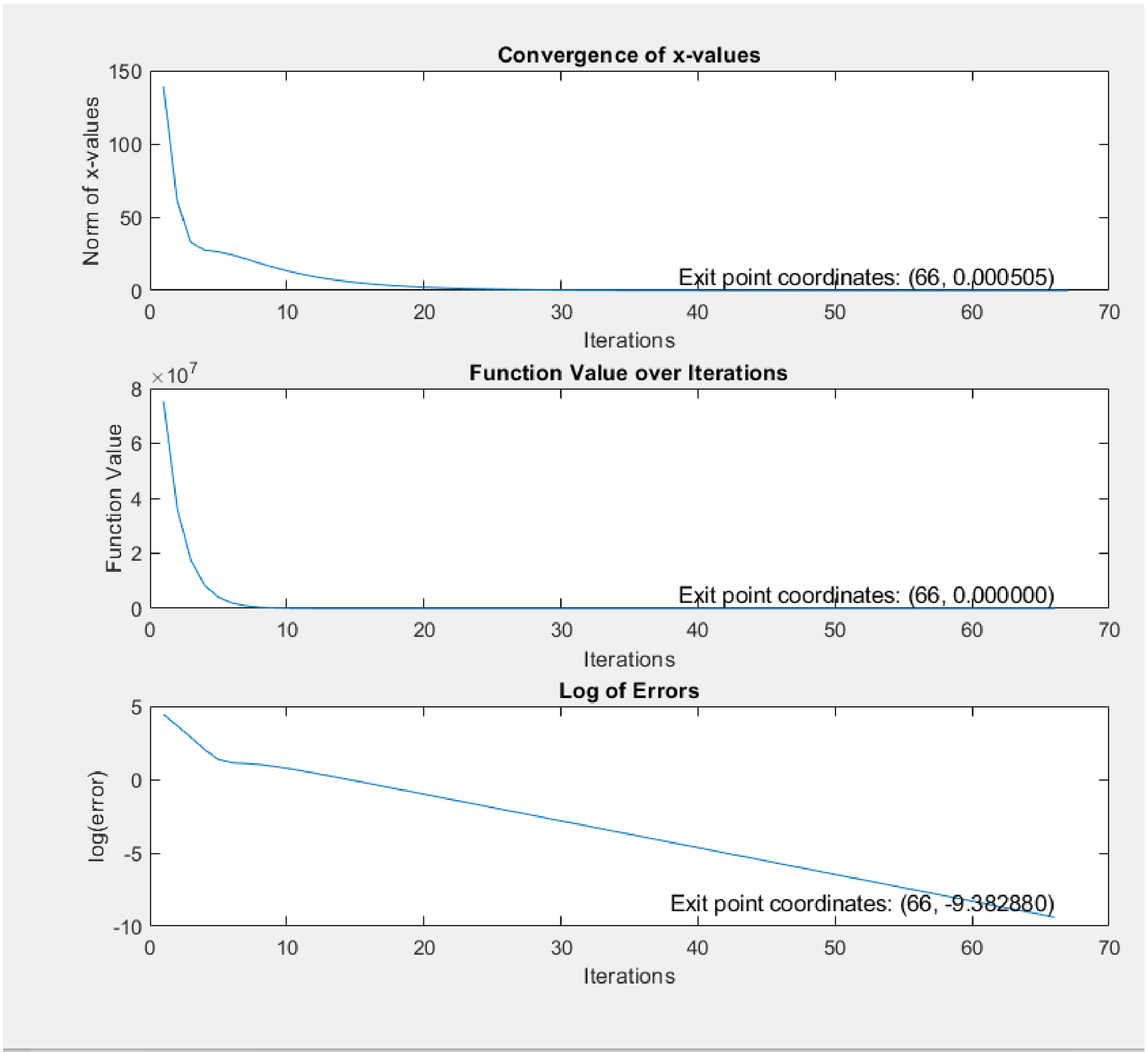


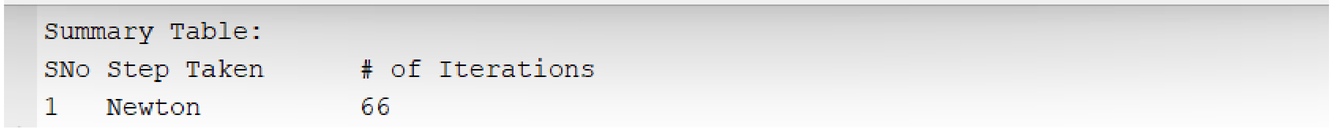


```
Summary Table:
SNo Step Taken       # of Iterations
1   Newton           61
```

Figure 9

For the initial condition $x0_{powell} = [56; -51; -44; 108]$, The root is approximately:
-0.000399059430
0.0000399059430
-0.000090000637
-0.000090000637

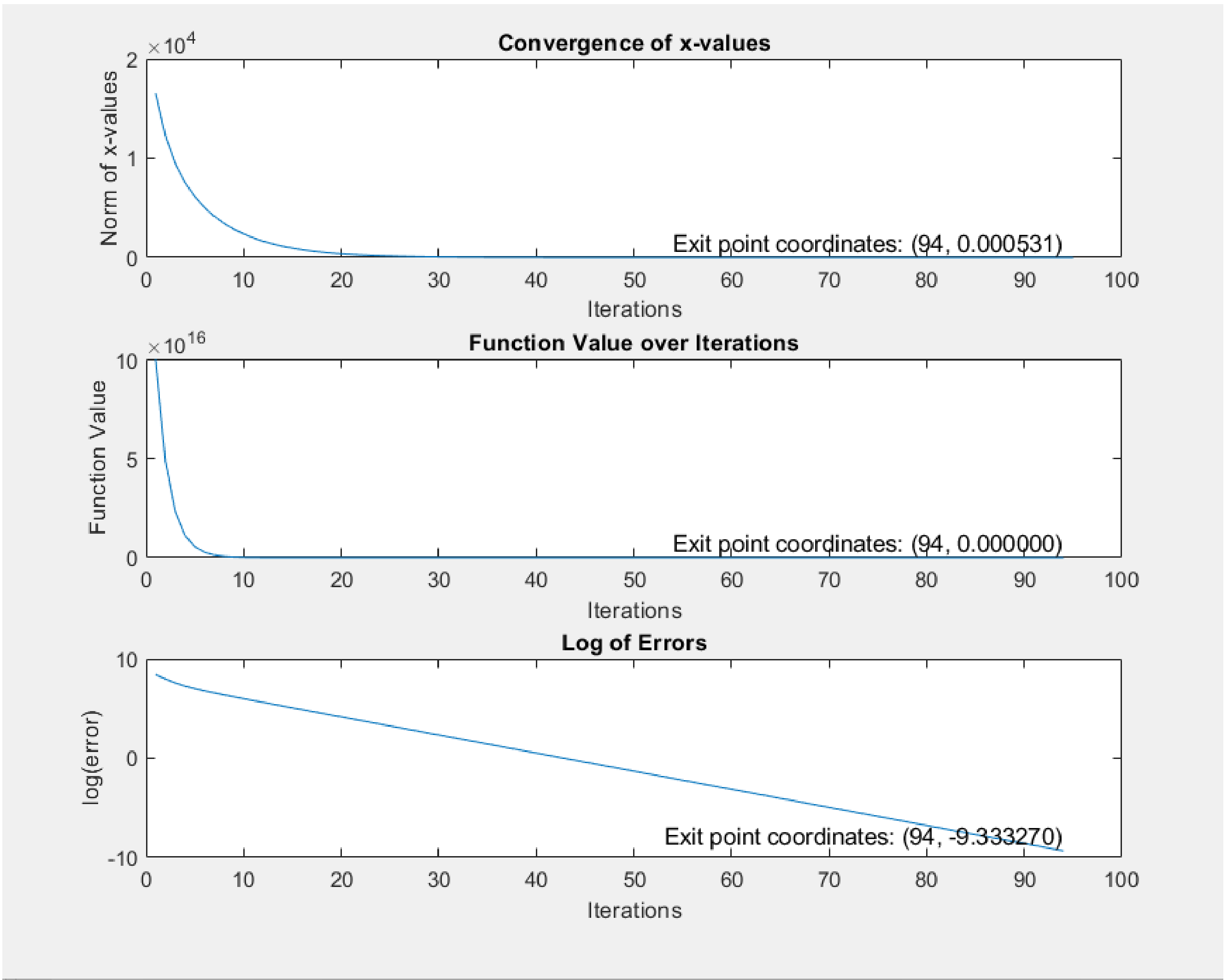


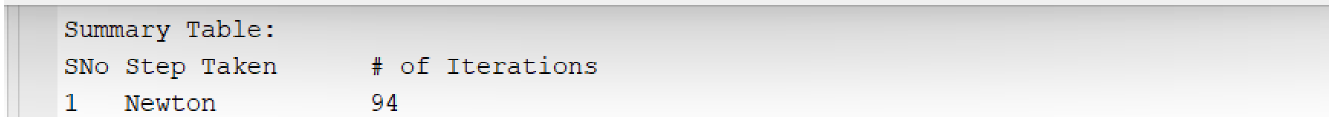


```
Summary Table:
SNo Step Taken       # of Iterations
1   Newton           66
```

Figure 10

For the initial condition $x0_{powell} = 5000 * [3; -1; 0; 1]$, The root is approximately:
0.00042922359278
-0.0000429223592
0.0000686757748
0.000068675774





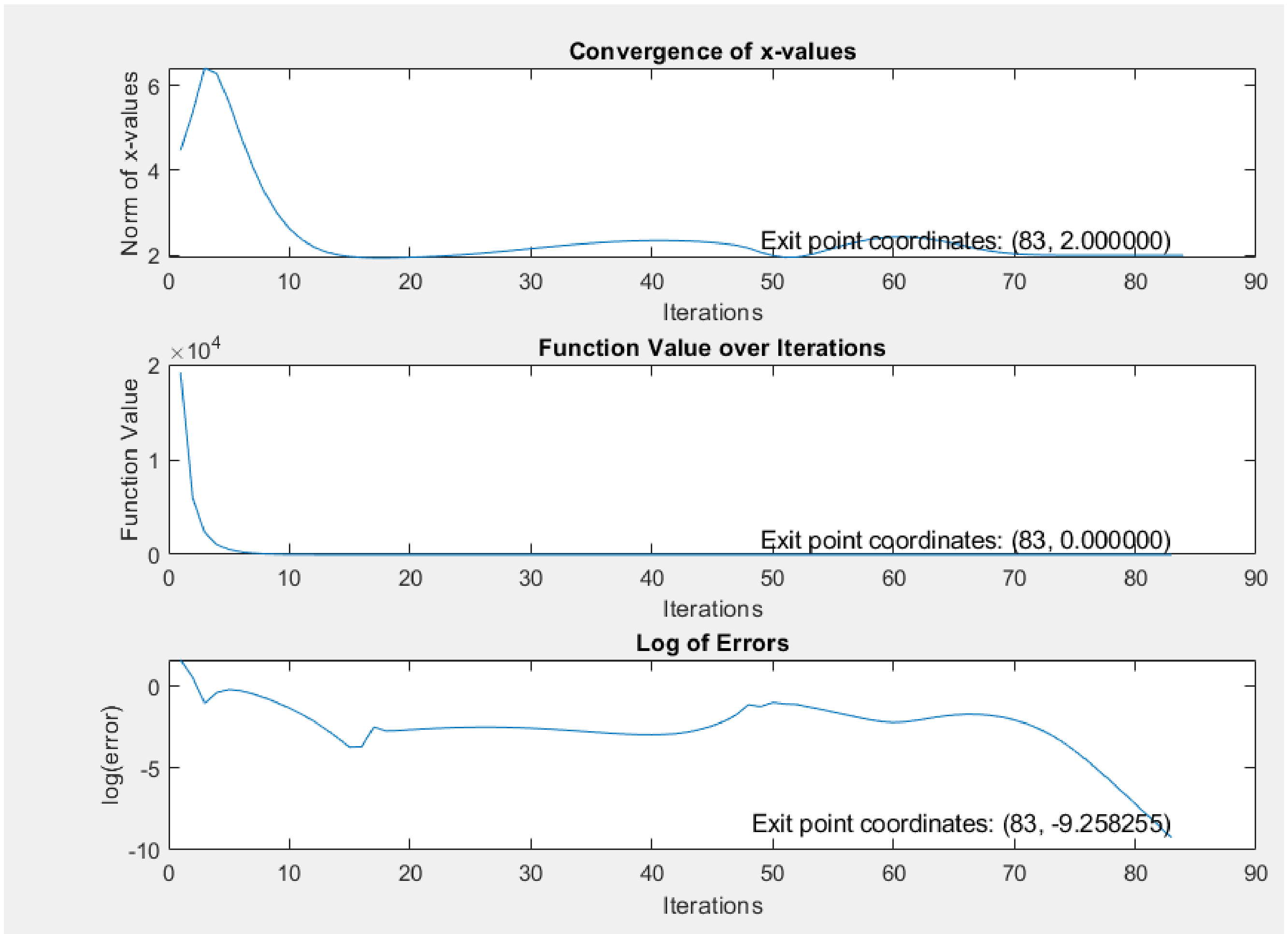
```
Summary Table:
SNo Step Taken       # of Iterations
1   Newton           94
```

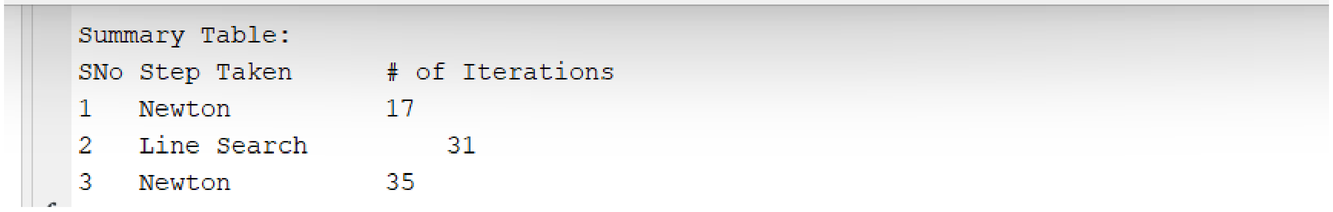
Figure 11

The rate of convergence for the Extended Powell Singular Function appears to be linear in all cases. As per my observation, the Powell function is not going into the Line Search or Trust Region updates and is continuing with the Newton step no matter what the initial conditions are, which is similar to the earlier case of the Rosenbrock function. An interesting observation is, that even for highly deviant initial conditions, the Powell function converges very fast as it can be seen that even with an initial point very far from the solution in the case of Figure 10, convergence was obtained. Also, in general, for the Powell function, initializations far from the solution took more number of iterations to converge.

2c

For Wood Function we have the following results
For the initial condition $x0_{wood} = [-3; -1; -3; -1]$, The root is approximately:
0.999970469581
0.999938970307
1.00003064915
1.0000596978

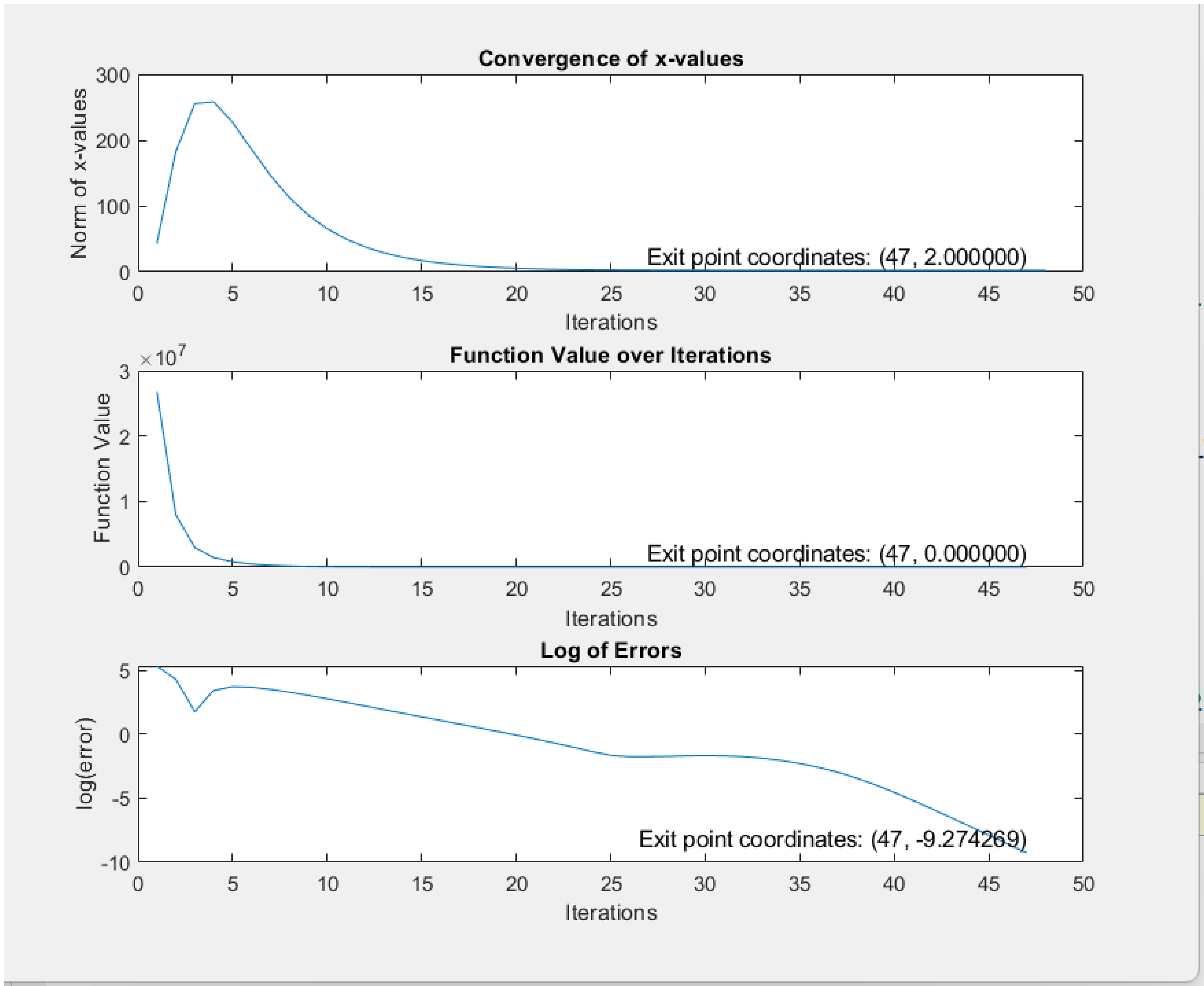




```
Summary Table:
SNo Step Taken       # of Iterations
1   Newton           17
2   Line Search          31
3   Newton           35
```

Figure 12

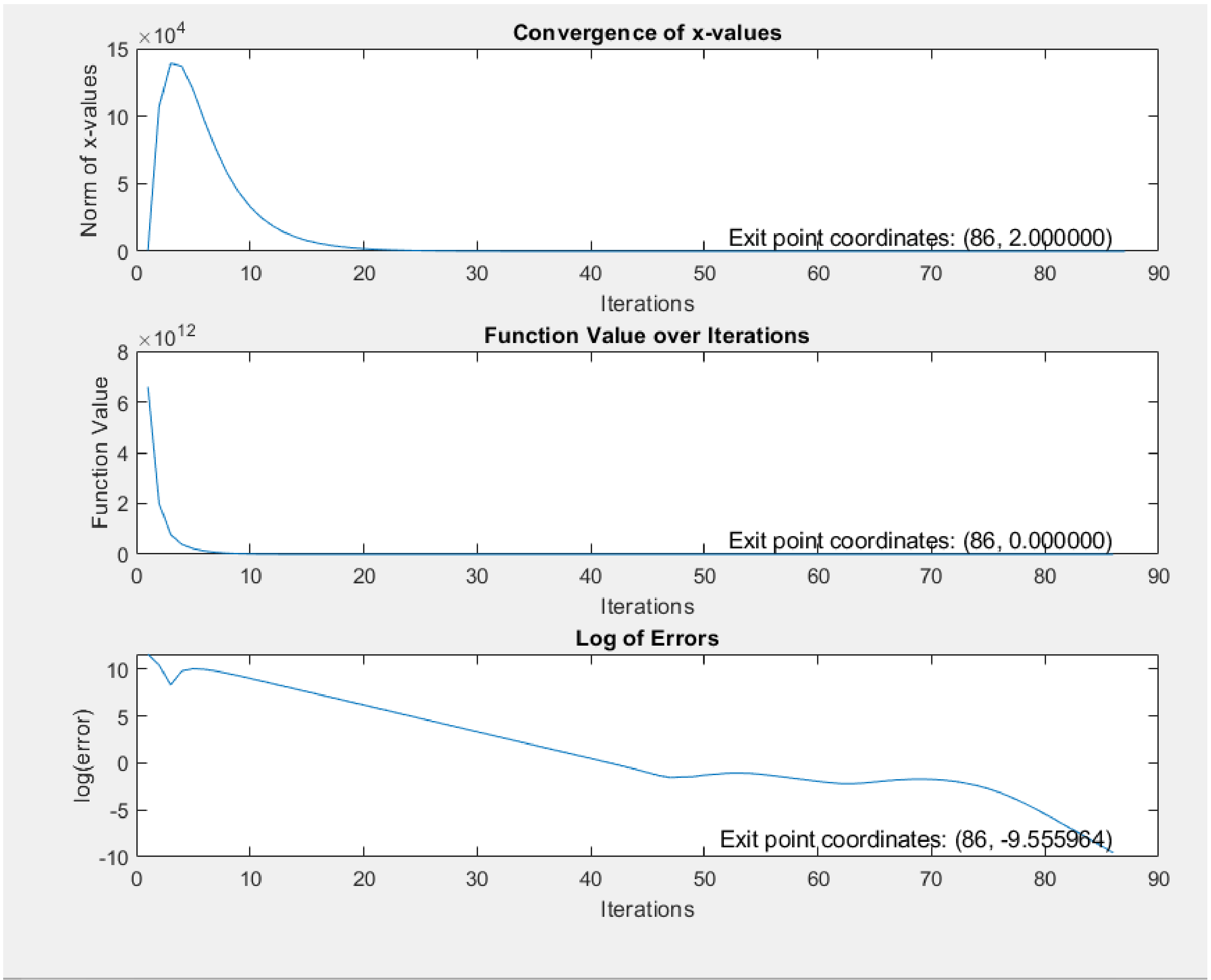
For Wood Function we have the following results
For the initial condition $x0_{wood} = [22; -31; -6; -19];$, The root is approximately:
1.0000303100
1.0000589606
0.99997105080
0.99994016762



```
Summary Table:
SNo Step Taken       # of Iterations
1   Newton           47
```

Figure 13

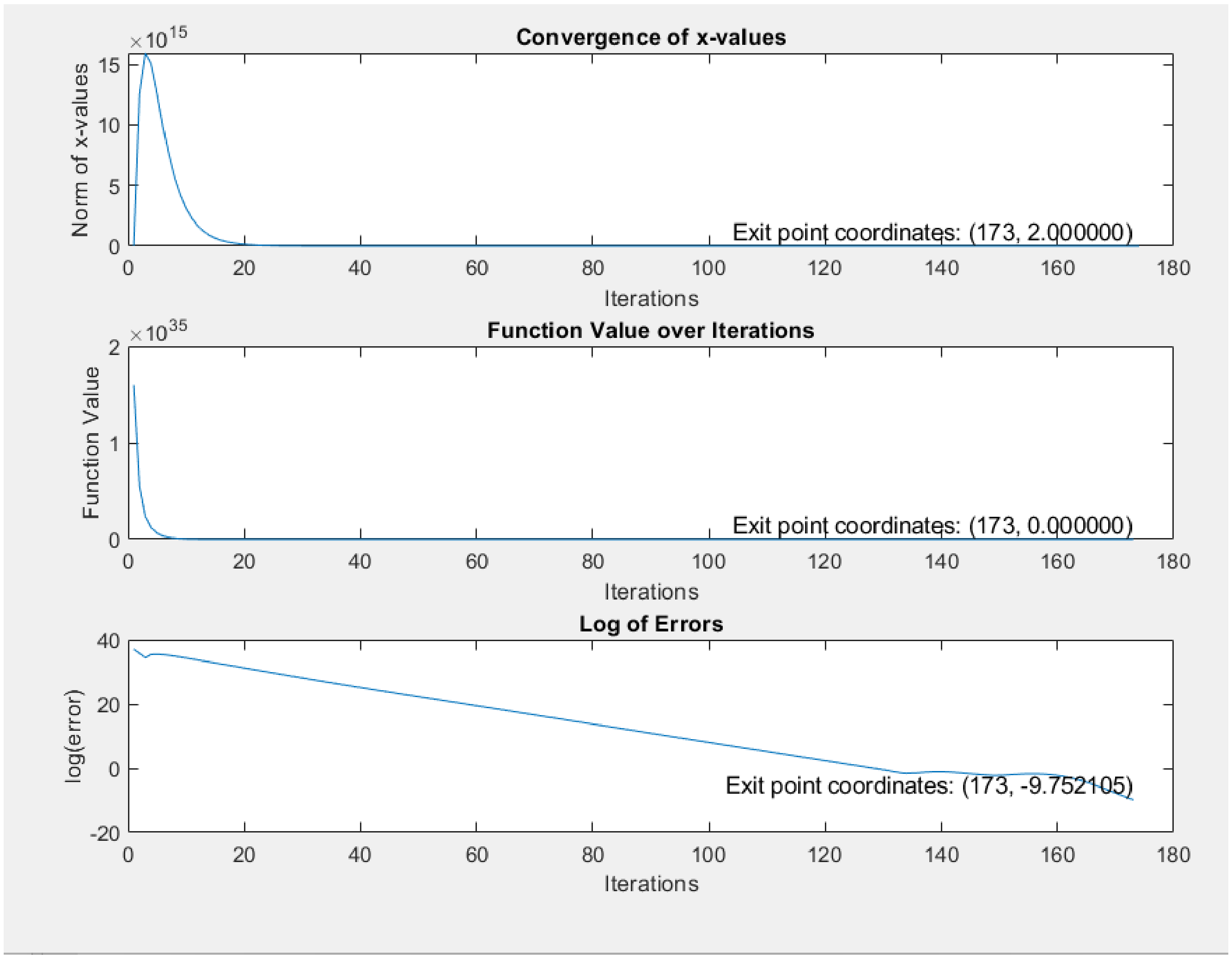
For Wood Function we have the following results
For the initial condition $x0_{wood} = [-507; 111; -77; 332]$, The root is approximately:
0.99997807750
0.99995469320
1.0000227532
1.0000443181

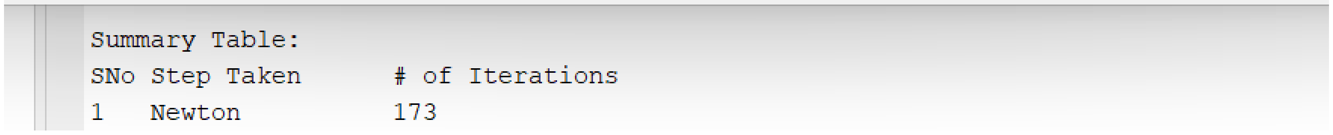


```
SNo Step Taken       # of Iterations
1   Newton           86
```

Figure 14

For Wood Function we have the following results
For the initial condition $x0_{wood} = 56784321 * [-3; -1; -3; -1]$, The root is approximately:
0.99998198383
0.99996276625
1.000018699
1.0000364212

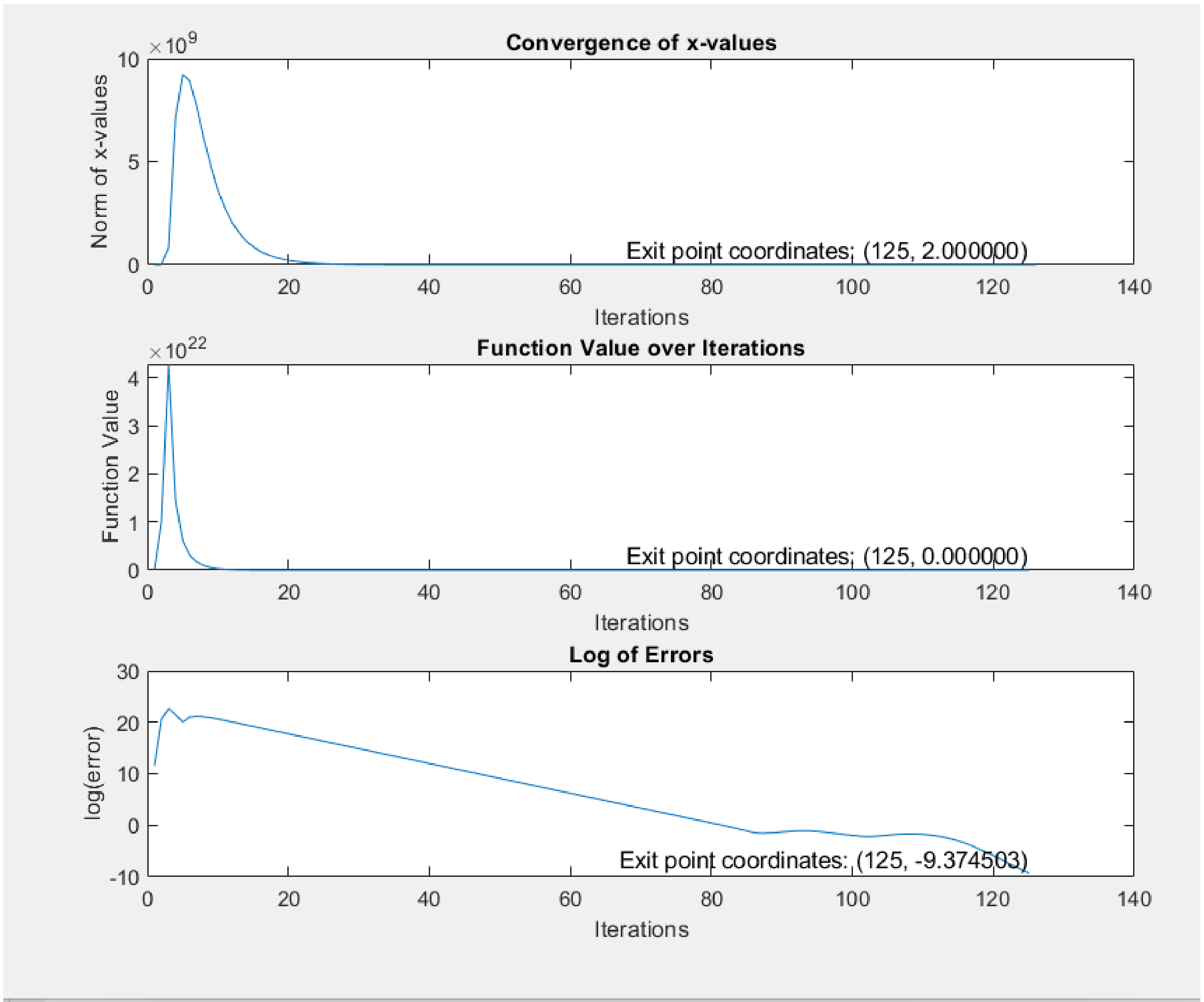


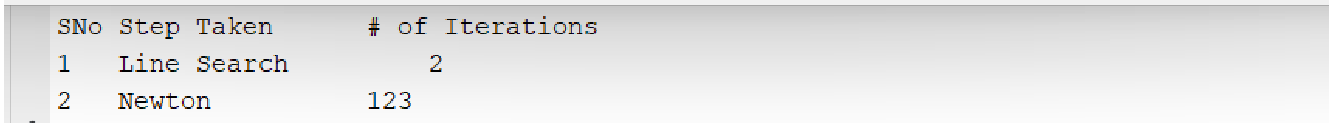


```
Summary Table:
SNo Step Taken       # of Iterations
1   Newton           173
```

Figure 15

For Wood Function we have the following results
For the initial condition $x0_{wood} = [0.5; 1.2; -1.3; -0.4]$, The root is approximately:
1.00002734700
1.00002734700
0.9999737713729
0.9999457654765





```
SNo Step Taken      # of Iterations
1   Line Search         2
2   Newton          123
```

Figure 16

The rate of convergence for the Wood Function appears to be linear for some number of initial iterations and then appears to transition into a quadratic. The Wood function successfully used line search for updates as can be seen in Figure 12. This behavior proves that the line search does work in the given code but needs complicated functions like the Wood function to be chosen.

Also, one more interesting observation is that the Wood function sometimes takes more iterations to converge for initializations with lesser deviation from the solution (Figure 12) than initializations that are far away from the solution (Figure 13). this might be because of the complex, highly non-linear landscape of the Wood function.