

Assignment 1

EE403 Unconstrained Optimisation

Date

10/11/2024

Team

Sinead McEvoy (21409002)

Logan Rave (21336546)

Louis Dwyer (21335163)

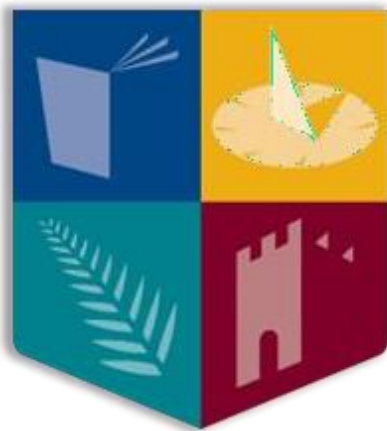
Cian Winder (21478641)

Facilitator

Dr Erivelton Nepomuceno

Thalita Emanuelle De Nazare

**Department of Electronic Engineering
Maynooth University**



**Maynooth
University**

National University
of Ireland Maynooth

Table of Contents

Table of Contents.....	2
Introduction.....	3
Part 1: Pseudo-code	4
1. Grid Search:	4
2. Fibonacci Method:	5
3. Quadratic Interpolation Method:.....	6
Comparison Between Methods	7
1. Computational Requirements	7
2. Speed of Convergence	9
3. Efficiency.....	9
Part 2: Graphical Analysis of Convergence.....	10
Method and Results	10
Conclusion	11
References.....	11
Appendix.....	12

Table of Figures

Figure 1: Grid Search method for a tolerance of 0.001.....	9
Figure 2: Fibonacci method for a tolerance of 0.001	9
Figure 3: Quadratic Interpolation method for a tolerance of 0.001	10
Figure 4: All methods for a tolerance of 0.001	10
Figure 5: Grid Search for a tolerance of 0.0001	11
Figure 6: Fibonacci method for a tolerance of 0.0001	11
Figure 7: Quadratic Interpolation method for a tolerance of 0.0001	12
Figure 8: All methods for a tolerance of 0.0001	12
Figure 9: Grid Search method for a tolerance of 0.01.....	13
Figure 10: Fibonacci method for a tolerance of 0.01	13
Figure 11: Quadratic Interpolation method for a tolerance of 0.01	14
Figure 12: All methods for a tolerance of 0.01	14

Introduction

“The procedure of finding the minimum of functions of one variable is an important element in any optimisation method.” This report describes three methods of solving unconstrained optimization problems: Grid Search, Fibonacci Search, and Quadratic Interpolation. These methods really represent different manners of navigating the search space for locating the optimum, with variable computational complexities, convergence speeds, and efficiencies.

In this assignment these three optimization techniques are implemented, performances compared, and behaviours analysed with graphs from MATLAB. The first part of this assignment contains the pseudo-code for each method and comparisons between the methods which include computational requirements, efficiency, and speed. Part 2 further explores these methods in MATLAB, seeking out the best optimization of a given function and analysing their convergence, number of iterations, accuracy, and computation efficiency. In this part, the most efficient method is identified.

Part 1: Pseudo-code

1. Grid Search:

```
// Grid Search Algorithm for Finding the Minimum of a Function f(x) over Interval [a, b]

Function: GridSearch(Function: f(x), Real: a, Real: b, Real: tol, Integer: divisions)
    // Step 1: Initialize search parameters and bounds.
    Set lower_bound = a
    Set upper_bound = b
    Initialize iteration_count = 0

    // Step 2: Loop until the interval [a, b] is within the tolerance level.
    While (upper_bound - lower_bound > tol)
        // Create grid points by dividing [a, b] into (divisions + 1) points.
        Define grid_points as an array of (divisions + 1) equally spaced points from lower_bound to
        upper_bound

        // Evaluate the function at each grid point.
        Define function_values as an array storing f(x) evaluated at each grid point

        // Find the grid point with the smallest f(x) value.
        Identify min_index as the position of the smallest value in function_values

        // Update interval bounds based on the location of the minimum:
        If min_index is the first position
            // Minimum is at the left end; adjust upper bound.
            Set upper_bound = grid_points[2]
        Else If min_index is the last position
            // Minimum is at the right end; adjust lower bound.
            Set lower_bound = grid_points[divisions]
        Else
            // Minimum is within the interval; narrow bounds around the minimum.
            Set lower_bound = grid_points[min_index - 1]
            Set upper_bound = grid_points[min_index + 1]
        End If

        // Increment the iteration count.
        Set iteration_count = iteration_count + 1
    End While

    // Step 3: Approximate the minimum as the midpoint of the final interval [a, b].
    Set x_gridmin = (lower_bound + upper_bound) / 2
    Calculate f_min as f(x_gridmin) // Evaluate f(x) at the approximate minimum

    // Step 4: Return the approximate minimum point and its function value.
    Return x_gridmin, f_min
End Function
```

2. Fibonacci Method:

```
// Fibonacci Method for Finding Minimum of f(x) over Interval [a, b]

Function: FibonacciMethod(Function: f(x), Real: a, Real: b, Real: tol)
    // Step 1: Initialize bounds, tolerance, and generate Fibonacci sequence.
    Set lower_bound = a
    Set upper_bound = b
    Generate Fibonacci sequence until the term n where (upper_bound - lower_bound) / Fibonacci(n) <
tol

    // Step 2: Define initial delta based on the n-th Fibonacci number.
    Set delta = (upper_bound - lower_bound) / Fibonacci(n)

    // Step 3: Calculate initial points xL and xR within the interval [a, b].
    Set xL = lower_bound + delta * Fibonacci(n - 2)
    Set xR = upper_bound - delta * Fibonacci(n - 2)

    // Step 4: Iterate through the Fibonacci sequence to narrow the interval.
    For k = 3 to (n - 1)
        // Evaluate f(x) at xL and xR.
        If f(xL) < f(xR)
            // If f(xL) < f(xR), update the interval [a, b] by moving the upper bound to xR.
            Set upper_bound = xR
            Set xR = xL // Shift xL to the right, becoming the new xR
            // Recalculate xL within the updated interval.
            Set xL = lower_bound + delta * Fibonacci(n - k - 1)
        Else
            // If f(xR) < f(xL), update the interval [a, b] by moving the lower bound to xL.
            Set lower_bound = xL
            Set xL = xR // Shift xR to the left, becoming the new xL
            // Recalculate xR within the updated interval.
            Set xR = upper_bound - delta * Fibonacci(n - k - 1)
        End If
    End For

    // Step 5: Estimate the minimum as the midpoint of the final interval [a, b].
    Set x_fibmin = (lower_bound + upper_bound) / 2
    Calculate f_min = f(x_fibmin) // Evaluate f(x) at the approximate minimum

    // Step 6: Return the approximate minimum point and function value.
    Return x_fibmin, f_min
End Function
```

3. Quadratic Interpolation Method:

// Quadratic Interpolation Method for Finding Minimum of $f(x)$ over Interval $[x_1, x_3]$

Function: QuadraticInterpolationMethod(Function: $f(x)$, Real: x_1 , Real: x_3 , Real: tol)

// Step 1: Initialize points and function values.

Set x_1 as the lower bound

Set x_3 as the upper bound

Set $x_2 = (x_1 + x_3) / 2$ // Midpoint between x_1 and x_3

Calculate $f_1 = f(x_1)$

Calculate $f_2 = f(x_2)$

Calculate $f_3 = f(x_3)$

// Step 2: Iterate until the interval width is within tolerance.

While ($x_3 - x_1 > tol$)

// Fit a quadratic function to points (x_1, f_1) , (x_2, f_2) , and (x_3, f_3) .

// Calculate numerator and denominator for the extremum formula.

Set numerator = $(x_2^2 - x_3^2) * f_1 + (x_3^2 - x_1^2) * f_2 + (x_1^2 - x_2^2) * f_3$

Set denominator = $(x_2 - x_3) * f_1 + (x_3 - x_1) * f_2 + (x_1 - x_2) * f_3$

// Calculate the new point x_{new} as the extremum of the fitted parabola.

If denominator is close to zero (near zero)

// If denominator is near zero, set x_{new} as the midpoint of the interval.

Set $x_{new} = (x_1 + x_3) / 2$

Else

// Otherwise, calculate x_{new} using the extremum formula.

Set $x_{new} = 0.5 * numerator / denominator$

End If

// Update bounds based on the position and value of x_{new} relative to x_2 .

If $x_{new} > x_2$

// If x_{new} is greater than x_2

If $f(x_{new}) < f_2$

// Shift the left bound to x_2 and update x_2 to x_{new} .

Set $x_1 = x_2$

Set $f_1 = f_2$

Set $x_2 = x_{new}$

Set $f_2 = f(x_{new})$

Else

// Otherwise, shift the right bound to x_{new} .

Set $x_3 = x_{new}$

Set $f_3 = f(x_{new})$

End If

Else

// If x_{new} is less than x_2

If $f(x_{new}) < f_2$

// Shift the right bound to x_2 and update x_2 to x_{new} .

Set $x_3 = x_2$

Set $f_3 = f_2$

Set $x_2 = x_{new}$

Set $f_2 = f(x_{new})$

Else

// Otherwise, shift the left bound to x_{new} .

Set $x_1 = x_{new}$

Set $f_1 = f(x_{new})$

End If

End If

End While

// Step 3: Approximate the minimum as the current x_2 .

Set $x_{min} = x_2$

Calculate $f_{min} = f(x_{min})$ // $f(x_2)$ is the approximate minimum function value

// Step 4: Return the approximate minimum point and function value.

Return x_{min}, f_{min}

End Function

Comparison Between Methods:

1. Computational Requirements

Grid Search Method

High: In every iteration, the grid search requires a function evaluation at a number of points throughout the grid. The computations increase directly with the number of divisions, hence being quite resource-consuming, especially if high accuracy is desired because then more and more grid points will be required for an adequate coverage of the search. [1] [2]

Fibonacci Search Method

Moderate: The Fibonacci Search has more computational efficiency compared to Grid Search because in every step, the former selects only two points according to the proportions of the Fibonacci sequence. This function evaluation technique is confined only to the key points that help effectively narrow down the interval. [3]

Quadratic Interpolation Method

Moderate to Variable: Quadratic Interpolation requires fewer evaluations in general since it uses previous point information to approximate the minimum of the function by a quadratic fit. However, each step is computationally heavier since it involves parabola fitting and solving for the extremum. [4]

2. Speed of Convergence

Grid Search Method

Slow: Grid search has a linear convergence rate since the interval width is reduced at a steady rate in each iteration. The grid search may converge slowly, particularly if high accuracy is required, because it must rely on evaluating uniformly spaced points rather than adapting to where the minimum of the function might be. [2]

Fibonacci Search Method

Moderate to Faster: The convergence rate is logarithmic. It systematically reduces the size of the interval using Fibonacci ratios, enabling the method to converge faster than Grid Search in smooth unimodal functions. [5]

Quadratic Interpolation Method

Fast (Superlinear): Quadratic Interpolation converges fast, especially when the function is quadratic or near quadratic in the interval. It has super-linear convergence because at each step it focuses on the vertex of the parabola; therefore, it is the fastest method for a smooth function that is quadratic or near quadratic around the minimum compared to the two previous methods. [6]

3. Efficiency

Grid Search Method

Low Efficiency: It is a rather inefficient technique as it evaluates points over a grid without adapting to the behaviour of the function. It is easy to implement but computationally wasteful especially for high-accuracy applications as it takes a very large number of evaluations. [7]

Fibonacci Search Method

Medium to High: Fibonacci Search is more efficient compared to Grid Search because it narrows the interval according to Fibonacci ratios, hence concentrating on the most promising areas, reducing unnecessary evaluations. A procedure that effectively balances simplicity and efficiency and works well for smooth, unimodal functions. [8] [9]

Quadratic Interpolation Method

High efficiency when favourable: Quadratic interpolation is efficient for functions that can be approximated by a parabola in the interval, given that this approach finds the minimum very quickly by using very few evaluations. Efficiency depends on the shape of the function itself, and it could be less effective if any irregularity in the function disturbs the parabolic approximation. [10]

Method	Computational Requirements	Convergence speed	Efficiency	Ideal Use
Grid Search	High	Slow	Low	Ideal for smaller data sets and fewer parameters
Fibonacci Search	Medium	Faster	Moderate to High	Unimodal functions on known interval
Quadratic Interpolation	Moderate (but varies)	Fast	High (if function is smooth)	Unimodal functions with clear minimum

Part 2: Graphical Analysis of Convergence

$$f(x) = x^4 + \frac{\sinh(x)}{\cosh(x)} \cdot \frac{1}{x}$$

Perform the following analyses:

1. Graphical analysis of convergence: implement each of the three methods to optimise the function and track how each method converges towards the minimum.

Tolerance of 0.001 for a range of [-4 4]:

8 Divisions for grid search.

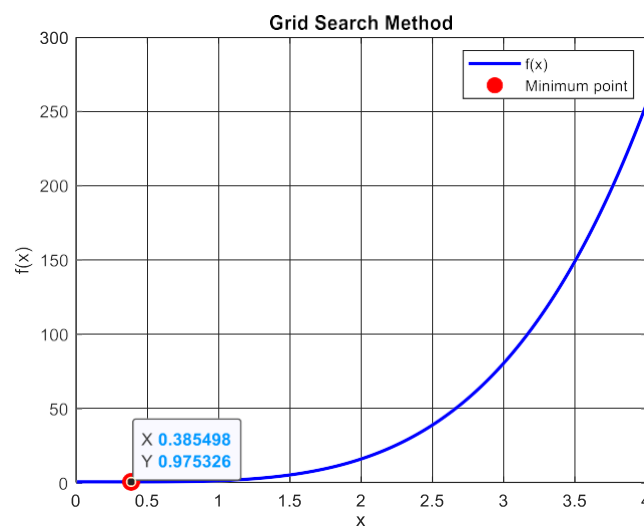


Figure 1: Grid Search method for a tolerance of 0.001

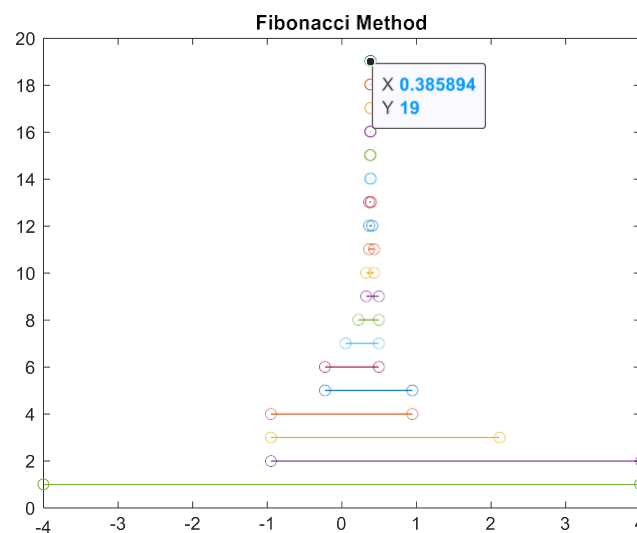


Figure 2: Fibonacci method for a tolerance of 0.001

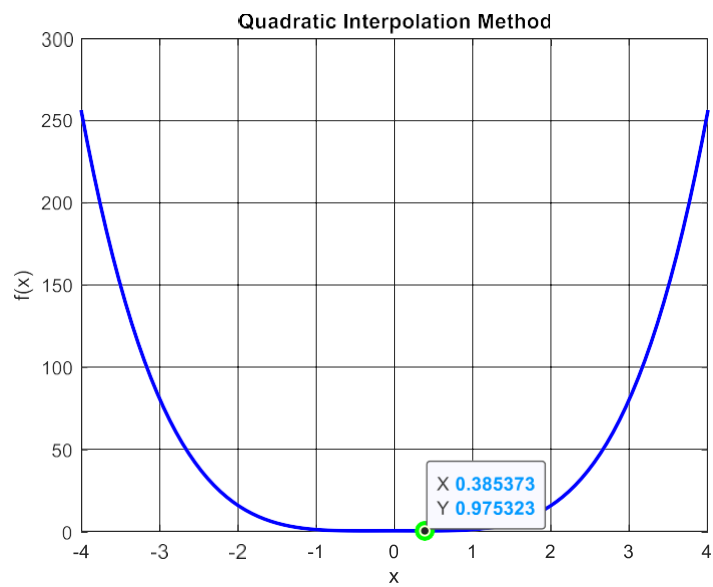


Figure 3: Quadratic Interpolation method for a tolerance of 0.001

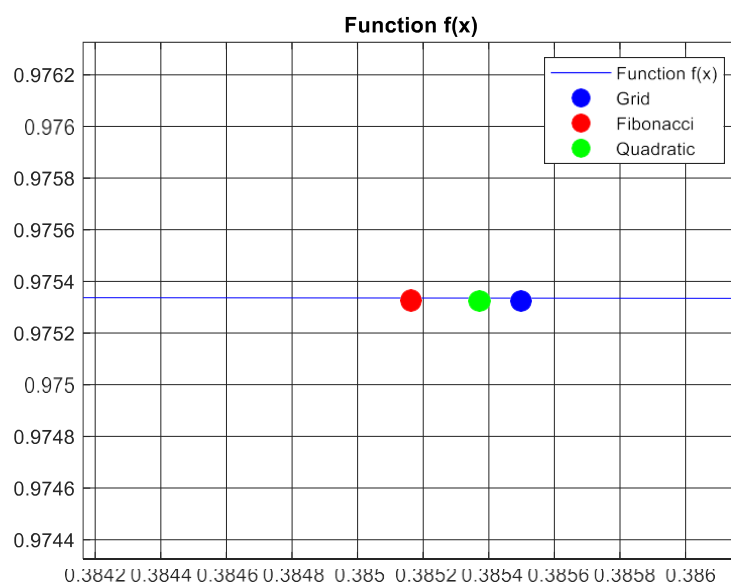


Figure 4: All methods for a tolerance of 0.001

“ Impact of initial conditions: vary the initial search intervals for each method and analyse how this affects their convergence behaviour. “

A 2nd more accurate tolerance of 0.0001 for the same range [-4 4] to compare the difference:

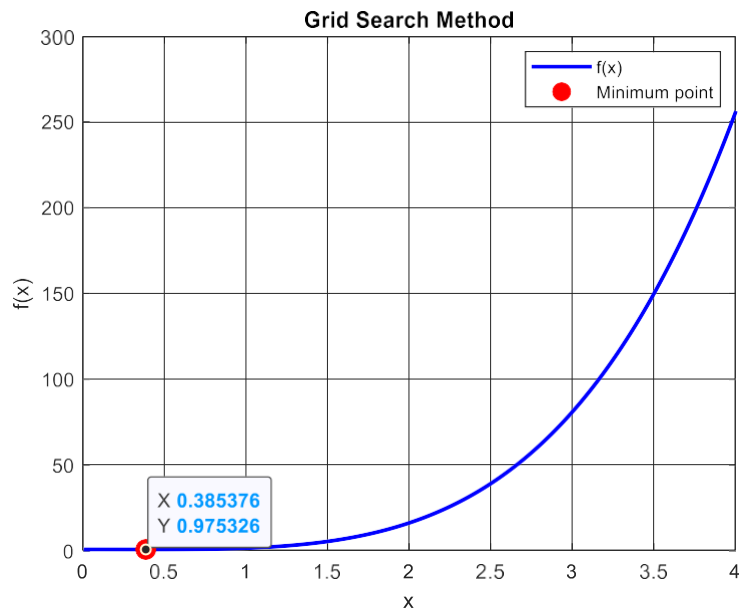


Figure 5: Grid Search for a tolerance of 0.0001

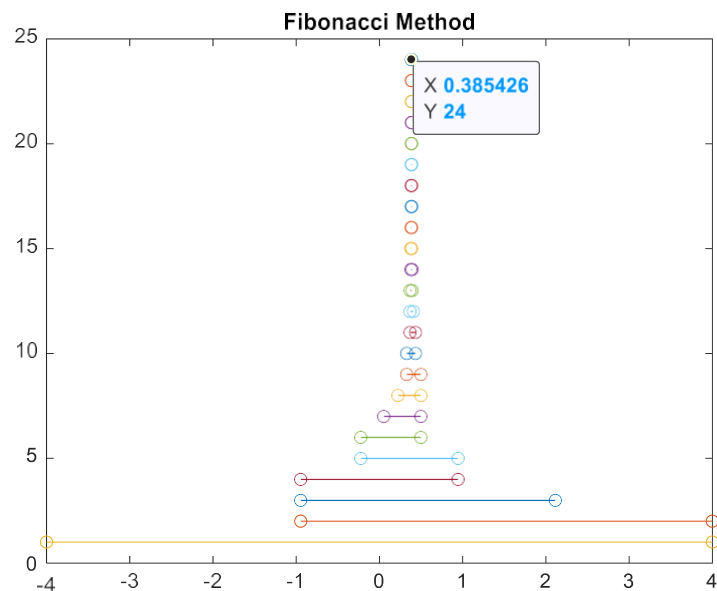


Figure 6: Fibonacci method for a tolerance of 0.0001

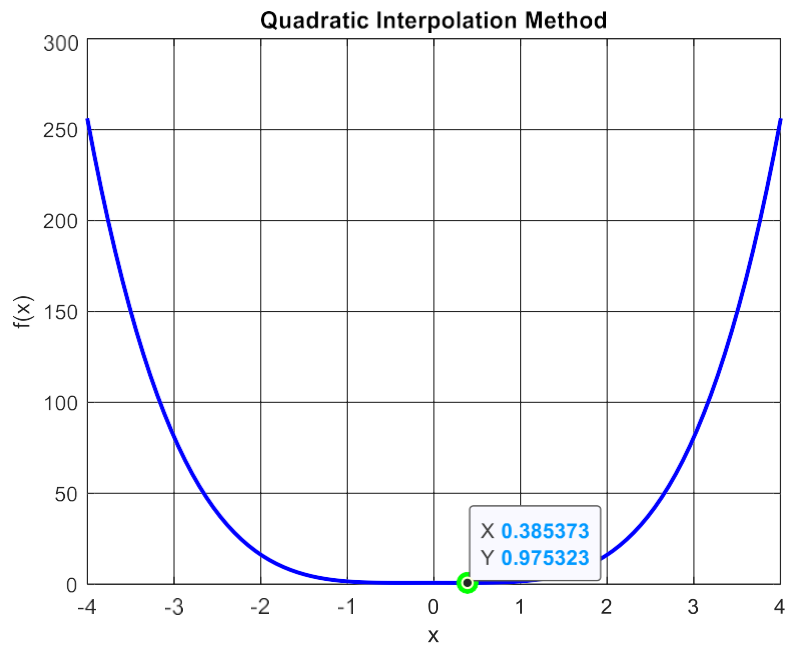


Figure 7: Quadratic Interpolation method for a tolerance of 0.0001

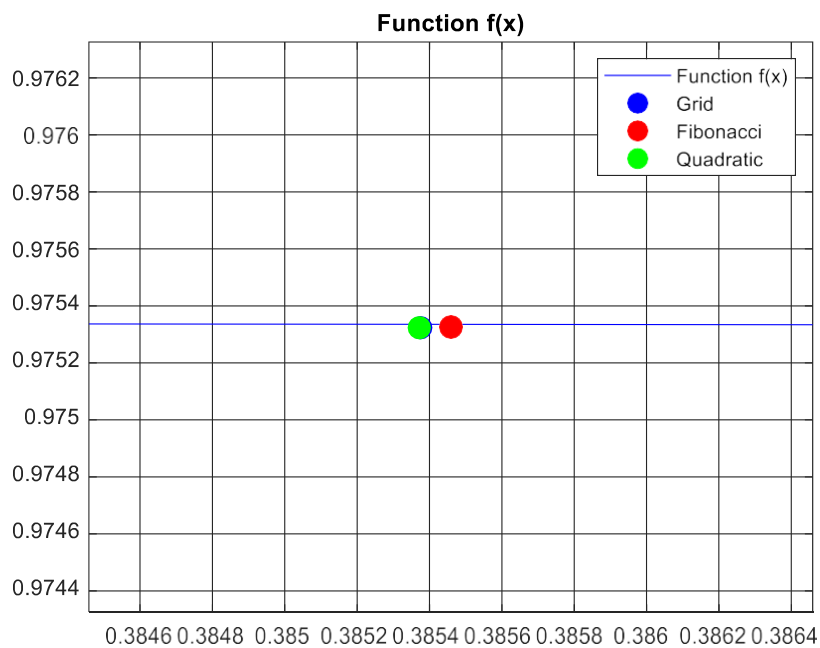


Figure 8: All methods for a tolerance of 0.0001

And a third tolerance of 0.01 for the range $[-4 \ 4]$

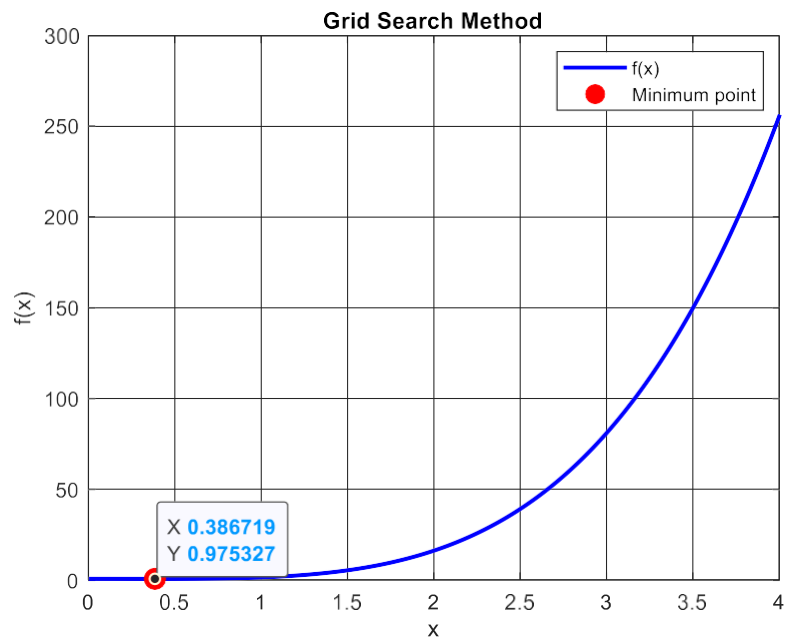


Figure 9: Grid Search method for a tolerance of 0.01

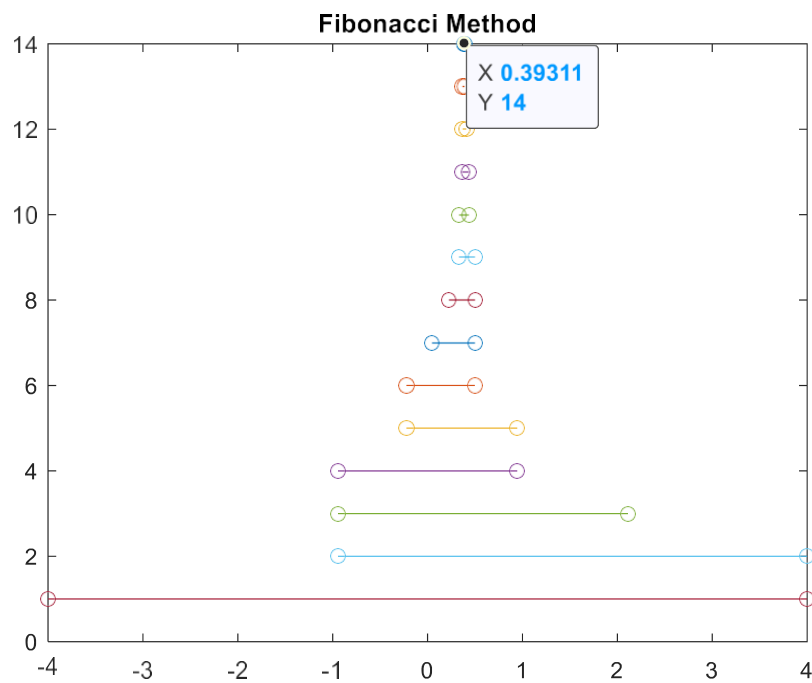


Figure 10: Fibonacci method for a tolerance of 0.01

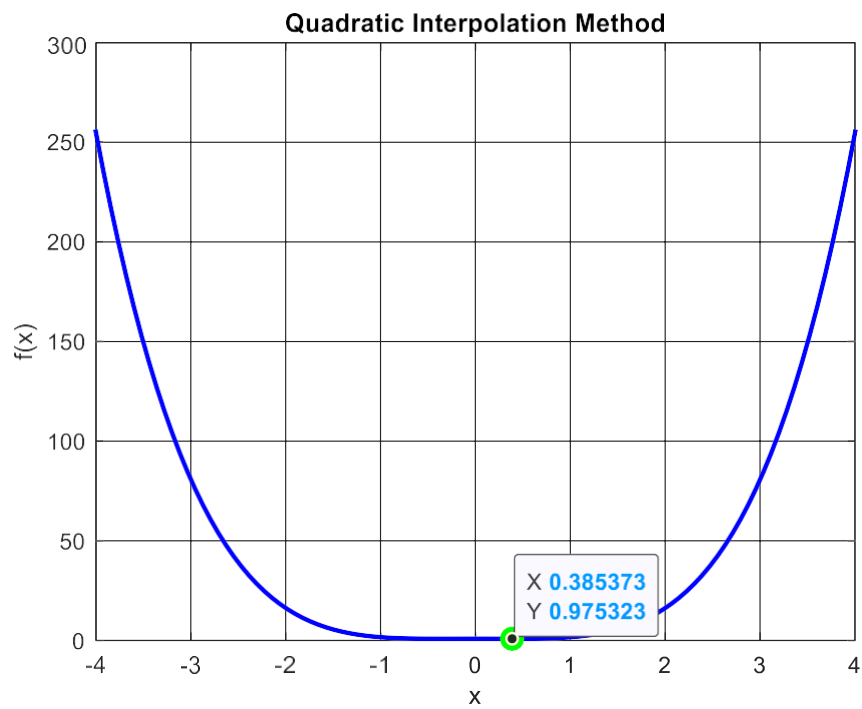


Figure 11: Quadratic Interpolation method for a tolerance of 0.01

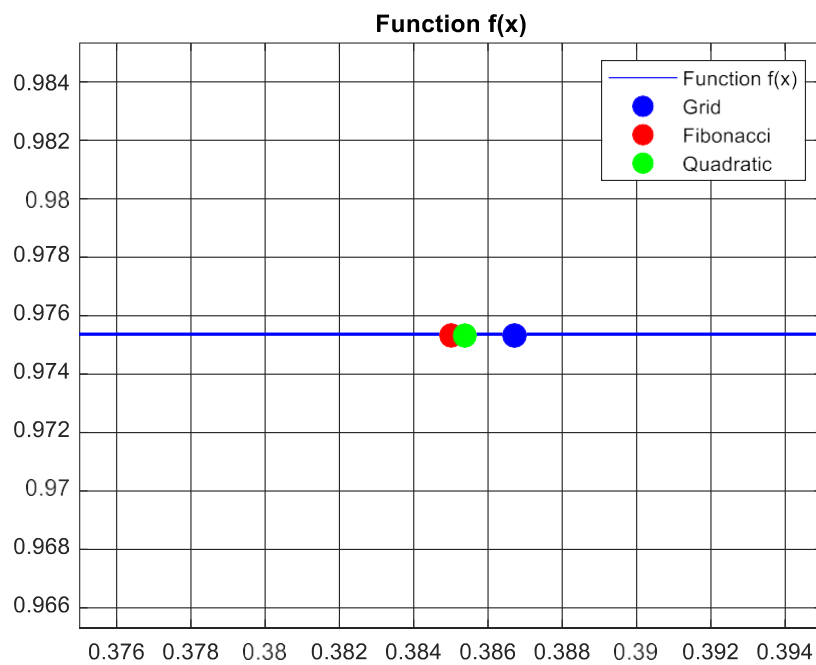


Figure 12: All methods for a tolerance of 0.01

Method and Results

Minimum Values recorded in Parts 2 and 3:

Method Used	Minimum Value for each tolerance					
	0.01		0.001		0.0001	
	x	y	x	y	x	y
Grid Search	0.386719	0.975327	0.385498	0.975326	0.385376	0.975326
Fibonacci	0.39311	0.97536	0.385894	0.97532..	0.385426	0.97532..
Quadratic Interpolation	0.385373	0.975323	0.385373	0.975323	0.385373	0.975323

2. Computational efficiency: compare the computational efficiency of the three methods by examining the number of function evaluations and iterations required.

Method Used	Iterations and Evaluations for each tolerance (Including initial Search)					
	0.01		0.001		0.0001	
	Iterations	Evaluations	Iterations	Evaluations	Iterations	Evaluations
Grid Search: (8 Divisions)	6	44	8	58	10	72
Fibonacci:	14	17	19	22	24	27
Quadratic Interpolation	26	28	29	31	33	35

Conclusion

As seen from the table above, the Grid Search method is proving to be the least computationally efficient, given that for a tolerance of 0.01 it takes 44 function evaluations to achieve a minimum value of 0.975327, compared to the Fibonacci method taking 17 evaluations to achieve a minimum value of 0.97536, and Quadratic Interpolation taking 28 evaluations to reach a minimum value of 0.975323. While Fibonacci is the most computationally efficient method in terms of function evaluations, it's the quadratic interpolation method that finds the smallest minimum value.

While the tolerance value becomes smaller and smaller, in the hopes of attaining a more optimal minimum value, the Grid Search method remains the least computationally efficient, however, the Fibonacci and Quadratic Interpolation method appear to be getting closer and closer in terms of function evaluations. This outcome falls in line with the research found in part 1 of the assignment, which states that while usually the most efficient option is the quadratic interpolation method, it may prove less efficient with complex functions that may disturb the parabolic approximation.

References

- [1] dreamio, “grid search,” dreamio, [Online]. Available: <https://www.dreamio.com/wiki/grid-search/#:~:text=Grid%20Search%20is%20a%20traditional,inefficient%20with%20high%2Ddimensional%20data..>
- [2] keylabs, “Hyperparameter tuning,” keylabs, [Online]. Available: <https://keylabs.ai/blog/hyperparameter-tuning-grid-search-random-search-and-bayesian-optimization/>.
- [3] wikipedia, “Fibonacci Search Technique,” wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Fibonacci_search_technique.
- [4] geeksforgeeks, “quadratic interpolation,” geeksforgeeks, [Online]. Available: <https://www.geeksforgeeks.org/quadratic-interpolation/>.
- [5] geeksforgeeks, “Fibonacci Search,” geeksforgeeks, 26 August 2024. [Online]. Available: <https://www.geeksforgeeks.org/fibonacci-search/>.
- [6] K. VANDEBOGERT, “METHOD OF QUADRATIC INTERPOLATION”.
- [7] R. N. Junior, “Model Optimization in Machine Learning: Random vs. Grid Search,” 28 January 2024. [Online]. Available: <https://www.linkedin.com/pulse/model-optimization-machine-learning-random-vs-neves-junior-phd-x459f/>.
- [8] R. R. S. Chauhan, “Fibonacci Search Algorithm: Description, Algorithm, and Complexity,” medium, June 23 2023. [Online]. Available: <https://medium.com/@rrschauhan210187/fibonacci-search-algorithm-description-algorithm-and-complexity-b7d450b17c23#:~:text=As%20a%20result%2C%20even%20in,the%20dataset%20is%20already%20sorted..>
- [9] wikipedia, “Fibonacci search technique,” wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Fibonacci_search_technique#:~:text=Fibonacci%20search%20is%20derived%20from,unimodal%20function%20in%20an%20interval..
- [1 S. W. Ibrahim, “Sabiha Walid Ibrahim,” 2024.
0]

Appendix

For this assignment, ChatGPT was used to provide extra information about each of the methods. Below are the prompts and responses:

Prompt:

“Please provide some information about computational requirements, convergence speed and efficiency for the methods grid search, Fibonacci search and quadratic interpolation”

Response:

Certainly! Below is a comparison of Grid Search, Fibonacci Search, and Quadratic Interpolation in terms of their computational requirements, convergence speed, and efficiency.

1. Grid Search

Computational Requirements:

- **Grid Search** is a brute-force method that involves dividing the search interval into a number of subintervals (grid points). At each grid point, the objective function is evaluated.
- **Time Complexity:** The number of function evaluations required grows linearly with the number of subintervals (i.e., grid points). If there are N divisions in the interval, the time complexity is $O(N)$, where each grid point requires one evaluation of the function.
- **Space Complexity:** Typically, $O(N)$ for storing the function values at each grid point (though you could use a more memory-efficient approach).

Convergence Speed:

- **Convergence:** Grid search has slow convergence, especially as the dimensionality of the search space increases. It works by systematically evaluating a coarse grid and refining the search within the region of interest. However, as the grid size gets finer, the search becomes more computationally expensive.
- **Rate:** The convergence rate is **linear** (or sometimes even slower) since the method does not exploit any structure of the function being minimized.

Efficiency:

- **Efficiency:** Grid search is computationally expensive because it requires evaluating the function at many points. The method does not exploit the shape or properties of the function and only works by testing a large number of values.
- **Best Use Case:** It's effective for finding an initial estimate for optimization problems but is inefficient for large-scale or high-dimensional optimization.

2. Fibonacci Search

Computational Requirements:

- **Fibonacci Search** is an efficient search technique that minimizes the number of function evaluations by dividing the interval using Fibonacci numbers.
- **Time Complexity:** The number of function evaluations required is proportional to the Fibonacci number $F(n)$, which grows exponentially. The number of function evaluations k required for a tolerance tol is approximately $O(\log(b - a) / \log(\phi))$, where ϕ is the golden ratio (~ 1.618). This is more efficient than grid search.
- **Space Complexity:** $O(1)$ since only a few variables are required to store the current bounds and the Fibonacci numbers being used to divide the interval.

Convergence Speed:

- **Convergence:** The convergence speed is significantly faster than grid search because Fibonacci search progressively reduces the search space by a factor related to the golden ratio.
- **Rate:** The convergence rate is **superlinear**, meaning it approaches the optimum faster than linear convergence.

Efficiency:

- **Efficiency:** Fibonacci search is relatively efficient, especially when compared to grid search, because it minimizes the number of function evaluations. It is particularly effective for unimodal functions.
- **Best Use Case:** It's efficient for one-dimensional optimization where the function is unimodal and the interval boundaries are known. Its computational efficiency makes it suitable for smaller-scale optimization tasks.

3. Quadratic Interpolation

Computational Requirements:

- **Quadratic Interpolation** uses a quadratic fit (a parabola) to approximate the function and find its minimum. It involves calculating the extremum (vertex) of the parabola formed by three points.
- **Time Complexity:** The number of function evaluations is $O(3)$ per iteration because it requires evaluating the function at three points to construct the parabola. Thus, for N iterations, the time complexity would be $O(N)$, but with significantly fewer evaluations compared to grid search or Fibonacci search.

- **Space Complexity:** $O(1)$, as only a few points and their function values need to be stored at each step.

Convergence Speed:

- **Convergence:** Quadratic interpolation can converge faster than Fibonacci search, especially if the function is smooth and unimodal, because it uses an approximation of the function to guide the search more directly to the optimum.
- **Rate:** The convergence rate is typically **quadratic**, which means the method converges very quickly once it's close to the optimal value, as each iteration provides a more accurate estimate of the minimum.

Efficiency:

- **Efficiency:** Quadratic interpolation is more efficient than both grid search and Fibonacci search for smooth unimodal functions. The method uses fewer function evaluations and, due to its quadratic convergence, can find the optimum with fewer iterations.
- **Best Use Case:** It is particularly efficient for unimodal, smooth functions, where it can quickly converge to the optimum. However, it may not be as effective if the function is non-smooth or highly oscillatory.