American University of Sharjah

MTH 343: Numerical Analysis I

Project Report

An Analysis of Muller's Method

Submitted By:
Taha Ameen

Student ID:
@00066555

Submitted To:
Dr. Marwan Abukhaled

November 29, 2019

Muller's Method

Root Finding Algorithm

Abstract

This document is a report that delineates my work in partial fulfilment of the requirements for the course MTH 343 - Numerical Analysis I. The work explores the history and utility of Muller's Method - a common root finding algorithm that uses quadratic interpolation to approximate the roots of a given function. This report explores the history and motivation behind the method before describing the algorithm. MATLAB based codes are provided to implement the method, and the results of the algorithm are also documented. Examples pertaining to the method are also illustrated.

Contents

List of Figures

1 Introduction

The objective of this report is to study Muller's Method with an emphasis on the analysis of its history and implementation. Muller's Method was introduced by David E. Muller in 1956 as a tool to find the roots of a given equation through numerical methods. Muller's Method was introduced as an advancement over existing numerical techniques inclusive of Newton's Method and Secant Method, and is inspired in principle by the Secant Method. Recall that the secant method is itself a modification of Newton's Method in which the derivative is approximated by the secant line. The secant line between two points is then used to determine its intersection with the x axis, which then serves as an update for the iterative guess that eventually converges to the root of interest. In contrast, Muller's Method uses three initial points and considers the parabola that contains them instead. This parabola can be found by means of quadratic polynomial interpolation.

The approach is useful because it is essentially a "higher order" version of the Secant Method. It uses a parabola to determine the intersection with the x axis instead of the straight line that the Secant Method uses. Subsequently, the intersection of the parabola with the x axis does a better job with respect to being closer to the actual root, than its linear counterpart. In order to better appreciate this method, we contrast it with Newton's Method and the Secant Method by first reviewing them and then exploring the advantages and disadvantages that Muller's Method has over them.

1.1 A Review of Preliminary Methods

Let $f : \mathbb{R} \to \mathbb{R}$ such that $\exists c \in \mathbb{R}$ for which $f(c) = 0$. We wish to find c. This section reviews two preliminary methods that motivate Muller's Method, namely Newton's Method and Secant Method.

1.1.1 Newton's Method

Newton's Method uses the Taylor expansion of f around c and then implements the linear approximation of the function to determine the following iterative formula.

Lemma 1.1. ***Univariate Newton's Method:*** *Let $f : \mathbb{R} \to \mathbb{R}$ and p_0 be an initial guess. Then*

$$p_{n+1} = p_n - \frac{f(p_n)}{f'(p_n)} \tag{1}$$

where $f'(x) = \frac{df}{dx}$.

Newton's Method is a fast algorithm that follows quadratic convergence (i.e is of $\mathcal{O}(h^2)$). Although the simplicity of the equation is attractive, there are certain limitations of this approach that are often stringent. For example, the algorithm only works when $\exists\ f'(p_n) \neq 0\ \forall n$. Further, if the initial guess p_0 is not sufficiently close to the actual root c, the algorithm easily diverges. Furthermore, the calculation of the derivative is not necessarily an easy task, primarily given the fact that the function f need not be an easily tractable object. This computational complexity is overcome by the Secant Method, which we now review.

1.1.2 Secant Method

Secant Method reduces the computational complexity associated with Newton's Method by eliminating the requirement for derivative computation. It relies on the following approximation.

$$f'(p_{n-1}) = \lim_{x \to p_n - 1} \frac{f(x) - f(p_{n-1})}{x - p_{n-1}} \tag{2}$$

If $x = p_n$ and p_{n-1} are sufficiently close so that $|f(p_n) - f(p_{n-1})| < \epsilon$, we can substitute (2) in (1) to obtain

$$p_n = p_{m-1} - f(p_{n-1}) \left[\frac{p_{n-2} - p_{n-1}}{f(p_{n-2}) - f(p_{n-1})} \right] \tag{3}$$

Clearly, the absence of derivative calculation speeds up the individual iterations, but at the expense of a loss in the order of convergence. There is a hierarchy of approximations here. Secant Method involves an approximation of the derivative, which itself is used in an approximation of the root. An interesting caveat worth mentioning is the requirement for two initial points in this method. This is best understood from the geometric interpretation shown in Fig. 1 [6].

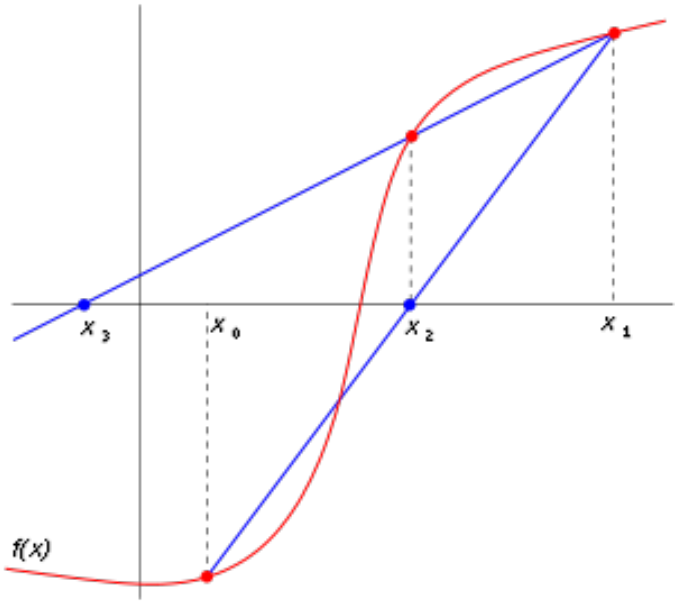

Figure 1: Secant Method: Geometric Interpretation

1.2 Why Muller's Method

The motivation for Muller's Method stems from the interplay of balance between the tradeoffs of Newton's Method and Secant Method [3]. Newton's Method has *quadratic convergence*, but each

iteration is computationally expensive. In contrast, Secant Method is computationally cheap to implement, but converges *slowly* compared to Newton's Method.

Muller's Method is an attempt to extract the advantages of both algorithms and find a middle ground with respect to both. It is an algorithm that converges faster than Secant Method, and is not as computationally expensive as Newton's Method. Having comprehended the requirement for the algorithm, we now study how it works.

Furthermore, Muller's Method has yet another advantage over its counterparts. It can be used to find complex roots of polynomials as well as other non-linear functions. Although other algorithms can be used as well, they require the initial guesses to be complex. This is a large degree of freedom over the initial guesses, and choosing a good one may be difficult if not impossible. However, Muller's method does not impose any such restrictions. It is possible to start off with three real guesses and end up with complex roots for the method.

2 Muller's Method: The Algorithm

Muller's Method is best summarized as an extension of the Secant Method that uses quadratic interpolation instead of linear interpolation. Therefore, instead of secant lines between two points, we use parabolas between three points. We first provide the raw algorithm before analyzing it [5].

Lemma 2.1. ***Muller's Method:*** *Let $f : \mathbb{C} \to \mathbb{C}$ be an analytic complex valued function. Let $p_0, p_1, p_2 \in \mathbb{C}$ be three initial guesses for $c \in \mathbb{C}$ such that $f(c) = 0$. For $n \geq 3$,*

$$p_n = p_{n-1} - \frac{2c}{b + \operatorname{sgn}(b)\sqrt{b^2 - 4ac}} \tag{4}$$

where

$$a = \frac{(p_{n-2} - p_{n-1})\left[f(p_{n-3}) - f(p_{n-1})\right] - (p_{n-3} - p_{n-1})\left[f(p_{n-2}) - f(p_{n-1})\right]}{(p_{n-3} - p_{n-1})(p_{n-2} - p_{n-1})(p_{n-3} - p_{n-2})}$$

$$b = \frac{(p_{n-3} - p_{n-1})^2\left[f(p_{n-2}) - f(p_{n-1})\right] - (p_{n-2} - p_{n-1})^2\left[f(p_{n-3}) - f(p_{n-1})\right]}{(p_{n-3} - p_{n-1})(p_{n-2} - p_{n-1})(p_{n-3} - p_{n-2})}$$

$$c = f(p_{n-1})$$

and $\lim_{n\to\infty} p_n = c$

2.1 A Derivation of the Formulae

Recall that we wish to find a parabola that fits the three points $(p_0, f(p_0)), (p_1, f(p_1)), (p_2, f(p_2))$. Since we are looking for a parabola, we begin by considering the quadratic polynomial without loss of generality [2].

$$P(x) = a(x - p_2)^2 + b(x - p_2) + c$$

Since the polynomial must pass through the aforementioned three points, it must satisfy the

following relations

$$f(p_0) = a(p_0 - p_2)^2 + b(p_0 - p_2) + c \tag{5a}$$
$$f(p_1) = a(p_1 - p_2)^2 + b(p_1 - p_2) + c \tag{5b}$$
$$f(p_2) = c \tag{5c}$$

Using the fact that $c = f(p_2)$, we have a system of linear equations in a and b that can be written as

$$\begin{bmatrix} (p_0 - p_2)^2 & (p_0 - p_2) \\ (p_1 - p_2)^2 & (p_1 - p_2) \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} f(p_0) - f(p_2) \\ f(p_1) - f(p_2) \end{bmatrix} \tag{6}$$

Solving for a and b gives the relations in (3). Hence, we have found a parabola that passes through the three points. Now, we are interested in the point where this parabola intersects the x axis. More rigorously, we wish to find p_3 such that $(0, p_3)$ lies on $P(x)$. But this is simply the root of a quadratic polynomial, which can be found using the quadratic formula. Precautionary measures taken to avoid round-off error problems due to the proximity of successive iteration results results in the following formula.

$$p_3 = p_2 - \frac{2c}{b + \operatorname{sgn}(b)\sqrt{b^2 - 4ac}} \tag{7}$$

Clearly, repeated application of this process results in the formula of Muller's Method. [1]

■

2.2 Application to Complex Valued Functions

An important advantage of Muller's Method is its applicability to complex valued functions as well. Let $f : \mathbb{C} \to \mathbb{C}$ be a complex valued function. We begin by assuming $p_0, p_1, p_2 \in \mathbb{R}$.

Firstly, it is worth noticing from (7) that $p_0, p_1, p_2 \in \mathbb{R} \not\Rightarrow p_3 \in \mathbb{R}$ due to the presence of $\sqrt{b^2 - 4ac}$.

Another possibility is that the function f is such that $a, b, c \notin \mathbb{R}$. For example, $f(x) = \sqrt{x+1}$ and $p_0 = -2 \implies f(p_0) \notin \mathbb{R}$. Therefore, since a, b depend on $f(p_0)$, there is no reason for $a, b, c \in \mathbb{R}$. This raises the question of what $\operatorname{sgn}(b)$ means when $b \in \mathbb{C}$. In such a case, if $b = x_b + iy_b$, where i is the imaginary unit, we define $\operatorname{sgn}(b) = \operatorname{sgn}(x_b)$. A drawback of this approach is that it does not account for the case when $b \in i\mathbb{R}$, i.e. when b is purely imaginary. However, such cases seldom arise as the imaginary parts are often the result of numerical inaccuracies in calculation.

2.3 Order of Convergence

The error analysis is not explored in immense detail, due to the restrictions of space in this project. It is worth mentioning however, that the order of convergence of Muller's Method is approximately $\mathcal{O}(h^{1.84})$ [4]. In fact, this value is obtained from the solution of $F(p) = p^3 - p^2 - p - 1 = 0$. This is interesting because the method is not quite as fast as Newton's Method, which is of $\mathcal{O}(h^2)$. However, the reliability and robustness of the algorithm combined with the reduction in computational complexity gives it an edge over Newton's Method. Muller's Method is much faster than the Secant Method.

3 Algorithm Implementation

3.1 Pseudocode

The following pseudocode is used to implement the algorithm.

Algorithm 1: Muller's Method

Inputs: $f, p_0, p_1, p_2, Tol, N_0$

Steps :

Find $f(p_i), i \in \{0, 1, 2\}$;
Find a, b, c using $f(p_i)$ by (3);
Initialize $j = 1$;
while $j \leq N_0$ *Or* $f(p_2) > Tol$ **do**
    Find p_3 using (7);
    for $i \in \{0, 1, 2\}$ **do**
        Set $p_i = p_{i+1}$;
    end
    $j = j + 1$;
end

3.2 MATLAB Implementation

Listing 1: MATLAB Code: Muller's Method.m

```
function [] = Mullers_Method(f,p0,p1,p2,Tol,N0)
%Mullers_Method Function to determine roots of f using Muller's
   Method
%          f : Function whose root is to be found (Input as String)
%  p0,p1,p2 : Initial Guesses
%       Tol : Tolerace
%        N0 : Maxmimum Allowed Iterations

f = inline(f); % Converts the string into a function

j = 1; %Initializing Loop Count
while j < N0 && abs(f(p2)) > Tol
    %% Finding a,b,c
    a = ((p1-p2)^1*(f(p0)- f(p2)) - (p0-p2)^1*(f(p1) - f(p2)))/((p0-
       p2)*(p1-p2)*(p0-p1));
    b = ((p0-p2)^2*(f(p1)- f(p2)) - (p1-p2)^2*(f(p0) - f(p2)))/((p0-
       p2)*(p1-p2)*(p0-p1));
    c = f(p2);

    %% Finding p3
```

```
    p3 = p2 - (2*c)/(b + sign(real(b))*sqrt(b^2 - 4*a*c)); %Accounts
        for Complex b

    %% Updating Guesses
    p0 = p1;
    p1 = p2;
    p2 = p3;
    j = j + 1; % Update Loop Count
end

%% Displaying Results
if abs(f(p2)) > Tol
    disp('Maximum Iterations Reached without reaching Tolerance.');
    disp(['Result: p2 = ' num2str(p2)]);
    disp(['Result: f(p2) = ' num2str(f(p2))]);
else
    disp(['Tolerance reached in ' num2str(j-1) ' Iterations']);
    disp(['Result: p2 = ' num2str(p2)]);
    disp(['Result: f(p2) = ' num2str(f(p2))]);
end
end
```

4 Results

In this section, we test Muller's Method by running the MATLAB algorithm and comparing the values with known results.
We examine the following equations

$$f_1(x) = x^2 - 2 \tag{8}$$
$$f_2(x) = x^2 + 16 \tag{9}$$
$$f_3(x) = x^5 - 4x^3 + x^2 + 3 \tag{10}$$
$$f_4(x) = \cos(x) - \sqrt{x+2} + x^{5/2} \tag{11}$$

The equations have been chosen to demonstrate the versatility of Muller's Method. We know the result that we expect for $f_1(x)$ is a real number, while for $f_2(x)$ it is a purely imaginary number. $f_3(x)$ is a higher order polynomial and $f_4(x)$ is a non linear equation that may or may not have roots.

We feed each of these functions into MATLAB. The results obtained are provided below. All four functions were tested with p_0, p_1, p_2 equal to $1, 1.5, 2$ respectively. The fast convergence of the method is particularly remarkable, as can be seen by the number of iterations required for the completion. Note that a maximum of 10 iterations and a tolerance of 10^{-3} was set for the program, as evident from the command line input entries. These are displayed in Fig. 2

An interesting observation regarding the method is its sensitivity to the selection of initial points. For example, choosing negative values for p_i in $f_1(x)$ gives the negative root. This is shown

```
>> Mullers_Method('x^2-2',1,1.5,2,0.001,10)
Tolerance reached in 1 Iterations
Result: p2 = 1.4142
Result: f(p2) = -4.4409e-16
>> Mullers_Method('x^2+16',1,1.5,2,0.001,10)
Tolerance reached in 1 Iterations
Result: p2 = 0+4i
Result: f(p2) = 0
>> Mullers_Method('x^5-4*x^3+x^2+3',1,1.5,2,0.001,10)
Tolerance reached in 4 Iterations
Result: p2 = 1.6521
Result: f(p2) = 9.0004e-06
>> Mullers_Method('cos(x) - sqrt(x+2) + x^(5/2)',1,1.5,2,0.001,10)
Tolerance reached in 3 Iterations
Result: p2 = 1.1218
Result: f(p2) = 3.1535e-06
```

Figure 2: Results of Muller's Method for 4 Different Functions

in Fig. 3. Another such case is seen in Fig. 4 where $f_3(x)$ is fed two different sets of initial points, and the result converges to two different roots. The roots in this case are complex in nature, and in fact are complex conjugates for the two cases shown in the algorithm.

To illustrate the importance of choosing good initial points, we explore what happens when initial points that are far away from the zeros are used. Figure 5 shows this scenario, where the algorithm does not converge.

```
>> Mullers_Method('x^2-2',-3,-1.5,-5,0.001,10)
Tolerance reached in 1 Iterations
Result: p2 = -1.4142
Result: f(p2) = -4.4409e-16
```

Figure 3: Example of Muller's Method's sensitivity to intitial points

```
>> Mullers_Method('x^5-4*x^3+x^2+3',-12,10,0,0.001,10)
Tolerance reached in 8 Iterations
Result: p2 = -0.33828-0.7602i
Result: f(p2) = -9.7347e-05-5.3606e-06i
>>
>> Mullers_Method('x^5-4*x^3+x^2+3',-4,2,2i,0.001,10)
Tolerance reached in 8 Iterations
Result: p2 = -0.33827+0.76019i
Result: f(p2) = -7.5159e-06-3.2035e-06i
```

Figure 4: Exaple of Muller's Method Converging to Different Roots

```
>> Mullers_Method('x^5-4*x^3+x^2+3',10^2,10^3,10^4,0.001,10)
Maximum Iterations Reached without reaching Tolerance.
Result: p2 = 465.3499+90.66449i
Result: f(p2) = 13695556385876.34+19650183971835.89i
>> Mullers_Method('cos(x) - sqrt(x+2) + x^(5/2)',100,1000,-100,0.001,10)
Maximum Iterations Reached without reaching Tolerance.
Result: p2 = -36.188+30.9006i
Result: f(p2) = 784723532105.0319-13126753376505.78i
```

Figure 5: Example of Muller's Method not Converging

5 Conclusion

Muller's Method is an iterative algorithm used to determine the roots of functions. These functions may be real valued or complex valued, and need not be restricted to any subspace of functions. Muller's Method combines the speed of Newton's Method with the computational simplicity of Secant Method, as it doe snot involves the finding of derivatives during the computation process. Consequently, it works wonderfully well and converges quickly. The method is capable of finding complex roots as well as real ones, and the convergence of the algorithm depends on the choice of initial points.

References

[1] Kendall E Atkinson, *An introduction to numerical analysis*, John Wiley & Sons, 2008.

[2] Richard L Burden and J Douglas Faires, *Numerical analysis, brooks*, Cole Pub **7** (1997).

[3] F Costabile, MI Gualtieri, and R Luceri, *A modification of muller's method*, Calcolo **43** (2006), no. 1, 39–50.

[4] Robin Kumar and Vipan, *Comparative analysis of convergence of various numerical methods*, Journal of Computer and Mathematical Sciences **6** (2015).

[5] David E Muller, *A method for solving algebraic equations using an automatic computer*, Mathematical tables and other aids to computation **10** (1956), no. 56, 208–215.

[6] Wikipedia, *Secant method - wikipedia page*, Wikipedia.