Programming Report 4

Executive Summary

In this program, we implement and test various non-linear root solvers. We do not run a larger parameterized set of experiments, but run a few directed tests for each problem. We want to test the convergence behavior for these methods on a few representative problems, and extrapolate general conclusions about the behavior of each method, and what kind of problems lead to which behavior. The observations will be presented with different runs of experiments and presented as evidence to answer individual questions and hypotheses we will be testing. We will explain our results in relevant conclusions about the capabilities of these algorithms.

Objectives

We will follow the task flow of the assignment, and conduct three tasks independently, though with much cross reference for the code implementation. The mathematics of these tasks is described as necessary, since most of this discussion is redundant in our implementation. We take the implementation of these methods from the class notes, and test numerical and theoretical results. Furthermore, for each task, we present representative problems with the appropriate results and plots. The methods we test are as follows:

- Regula Falsi Method
- Secant Method
- Newton's Method
- Modified Newton's Method
- Steffensen's Method

We have the following objectives we want to test:

1. **Higher Order Roots**: We test the various methods for different multiplicities of a single root, and we use a simplified problem for this purpose. We also change the various parameters for each method and a run a small number of experiments for each set of starting conditions, and display and discuss the results.
2. **Distinct Roots and Trends as They Coalesce**: We use distinct roots and discuss the conditions where the roots coalesce. This is equivalent to approaching different multiplicities in an approximate fashion, and allows use to test our hypotheses regarding multiplicities more comprehensively. We transform our test function slightly for this purpose. We divide the problem into three parts: distinct roots, scaling and then finally coalescing.

You may refer to each of the above descriptions as necessary.

Note on Implementation

We implement the code on python using a Jupyter notebook. This allows us to use different code blocks for each individual task, and also allows us to conduct each kind of test individually, for each task. We can also conduct interactive testing and display each of these tests separately. The outputs and the tests are displayed after each test/task and can be run independently. We use the NumPy library to construct and use data structures, and various other libraries for error analysis, testing, and visualization (NumPy, SciPy, Pandas, and Matplotlib). We will cite our usage of libraries wherever they have been used. Most relevant plots will be included in this report. Certain parts of the experiments automatically save graphs, tables, and plots in the current directory, and this functionality can be toggled with a Boolean variable. This should be kept in mind when running the source files on your own machine!

Higher Order Roots

Consider problems of the form $f(x) = (x - \rho)^d$, i.e., a single root at ρ of multiplicity d.

0.1 Using Standard Newton's Method

Use standard Newton's method, i.e., $m = 1$, and empirically demonstrate the convergence rate trends as you take $d = 2, 3, \ldots$. You should demonstrate the behavior with two to three different x_0 for each d and record your observations on the behavior.

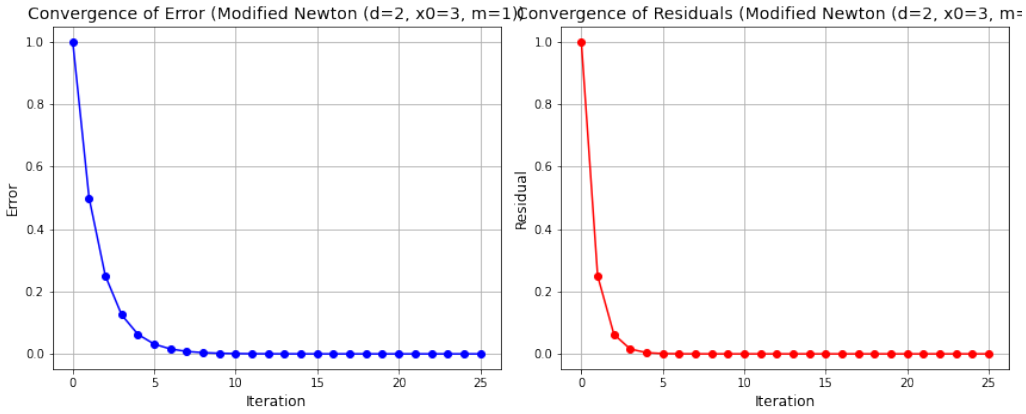
The first experiment has m=1 (Standard Newton) where for each experiment we take three values of the initial guess

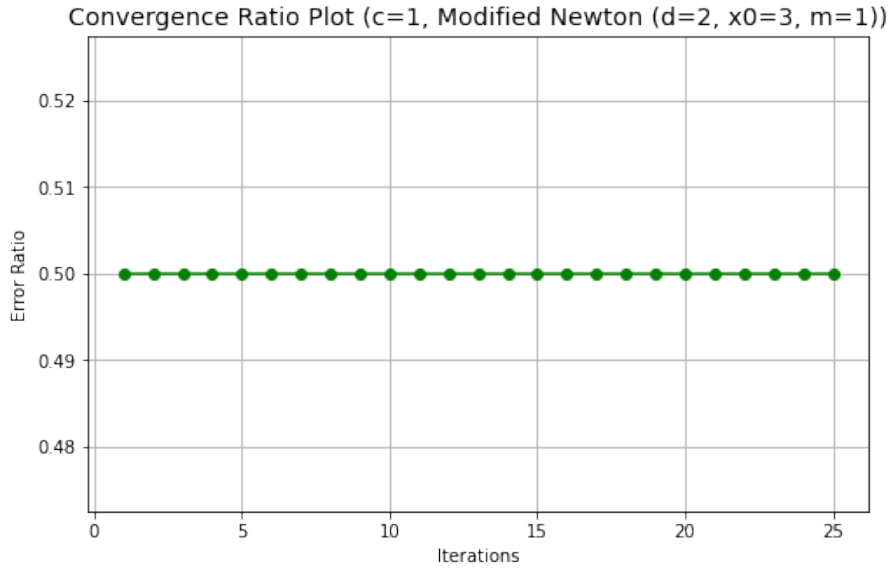
$$x_0 = r + 1, r + 5, r + 20$$

The first experiment gives us the following result:

Parameter	Value
Initial Guess	3.0
Multiplicity (m)	1.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	3	2	1	2.98e-08	8.88e-16	25

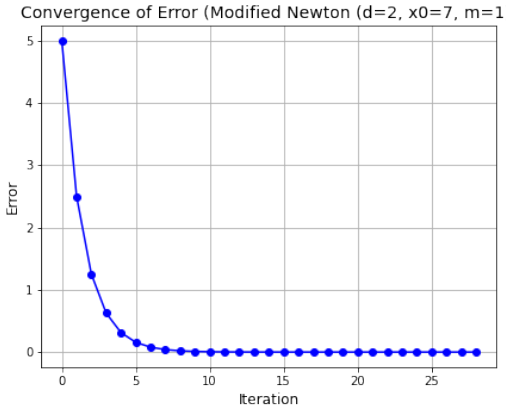


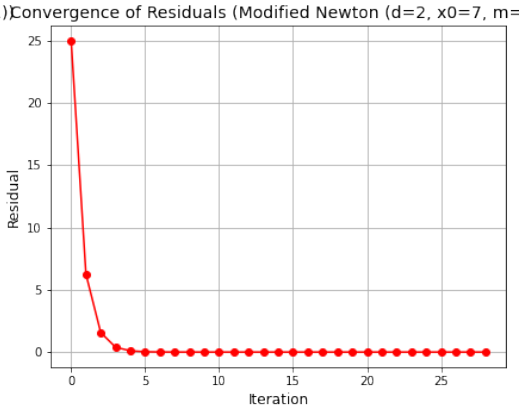


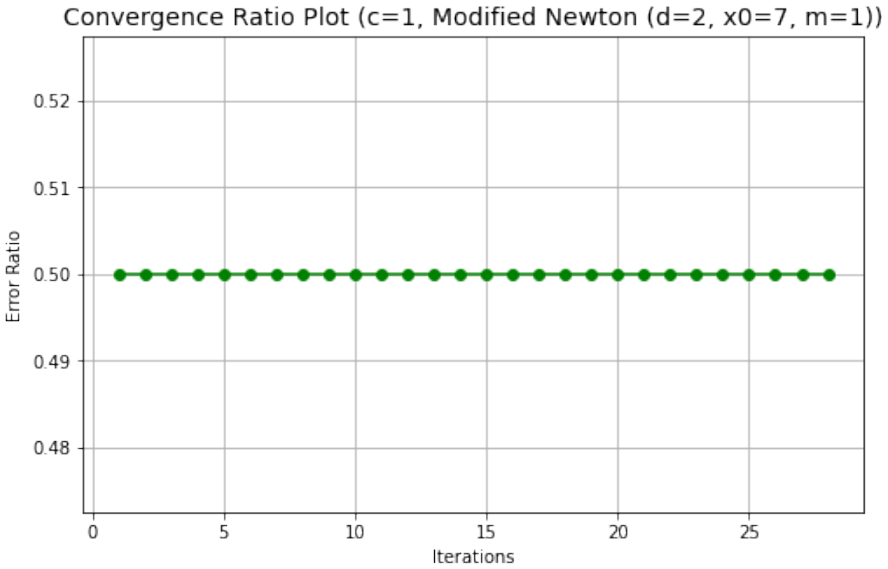
The second experiment we take out initial guess to be further away (r+5):

Parameter	Value
Initial Guess	7.0
Multiplicity (m)	1.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	7	2	1	1.86e-08	3.47e-16	28



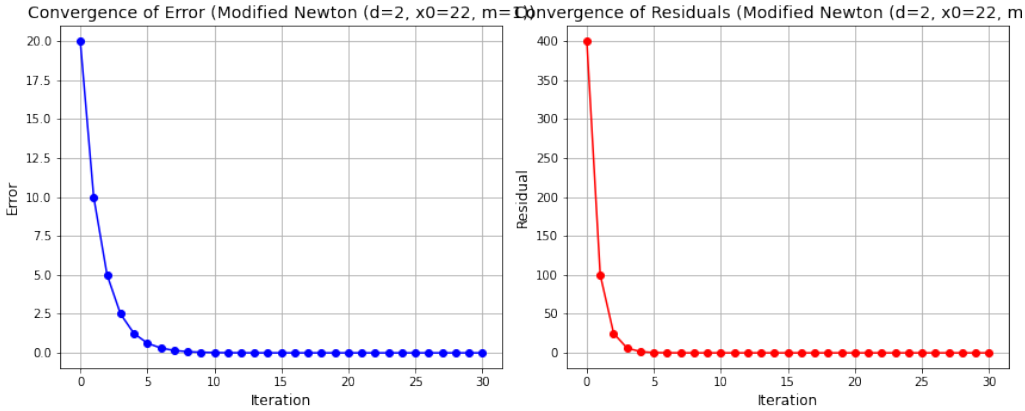


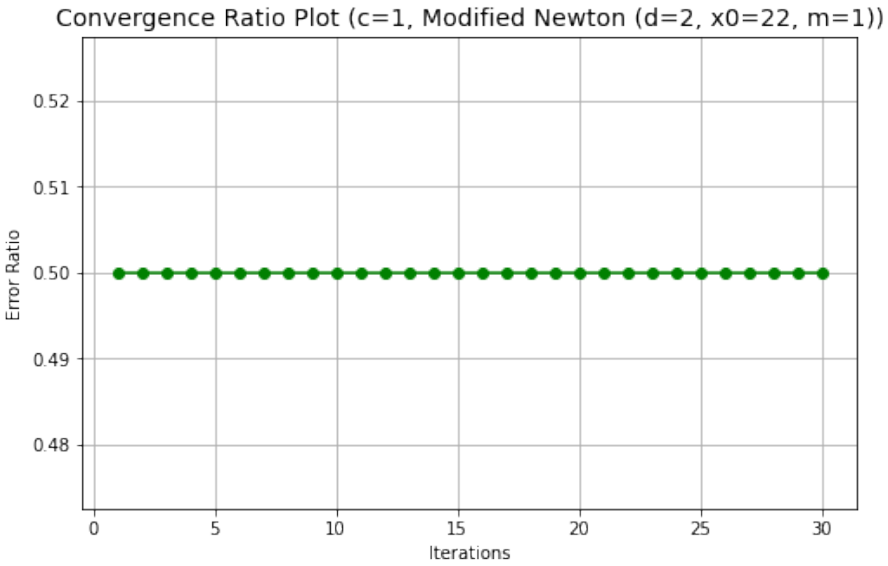


For the third we take an even further guess:

Parameter	Value
Initial Guess	22.0
Multiplicity (m)	1.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	22	2	1	1.86e-08	3.47e-16	30



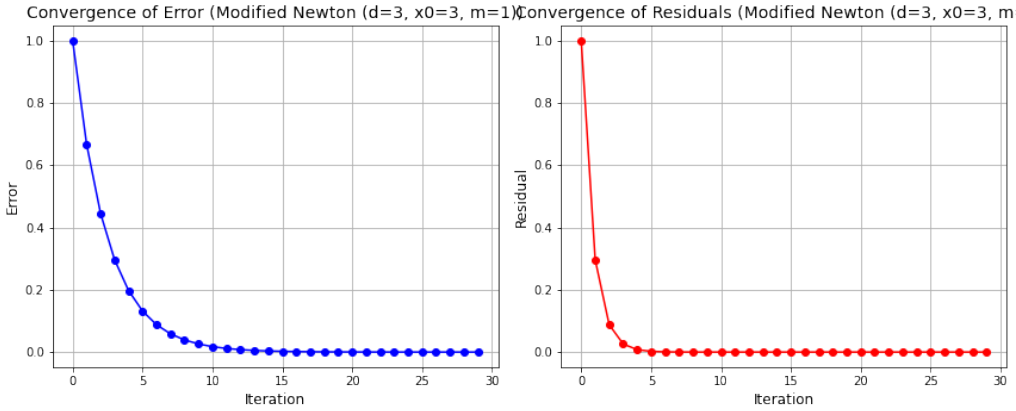


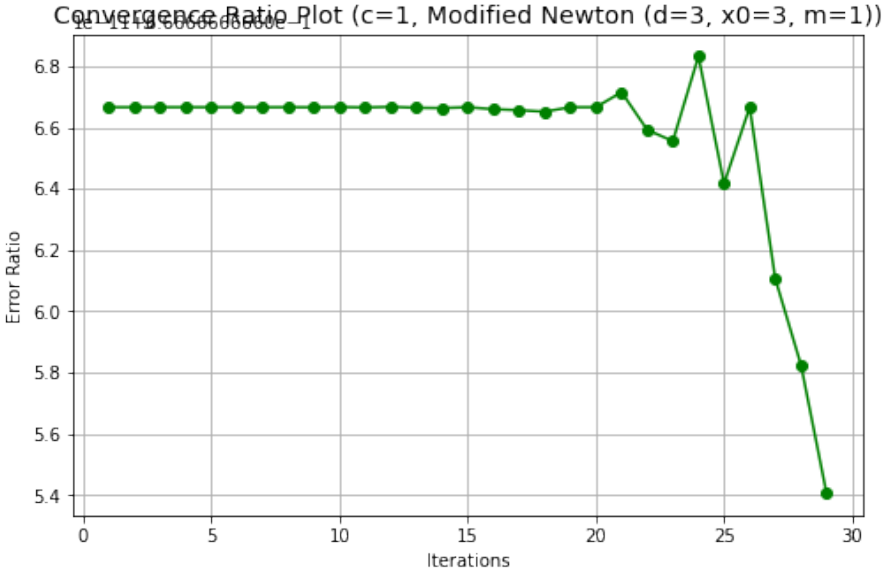
Conclusion: We get linear convergence in these cases, and the methods become slower (more iterations) as our guesses move away from the root. This is expected behavior.

We now test for higher and higher multiplicities. Note the multiplicity in the table represents the parameter for the method. The multiplicity for the root (d=3) is shown in the plots:

Parameter	Value
Initial Guess	3.0
Multiplicity (m)	1.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

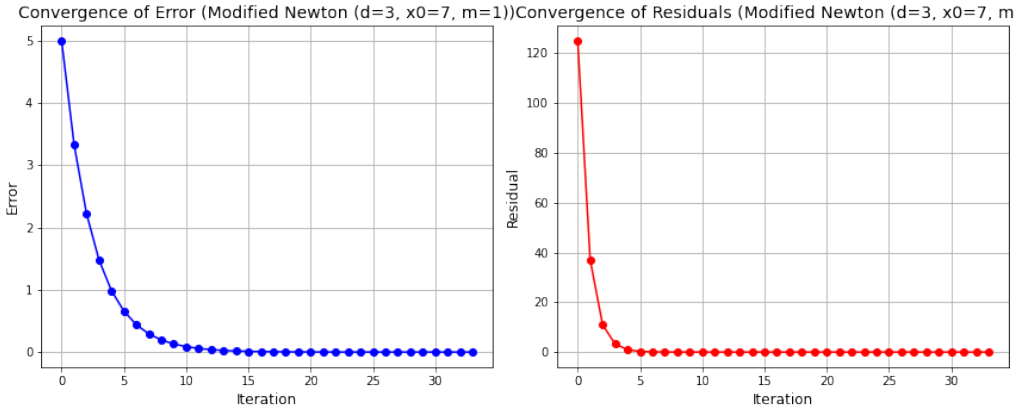
Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	3	2	1	7.82e-06	4.79e-16	29

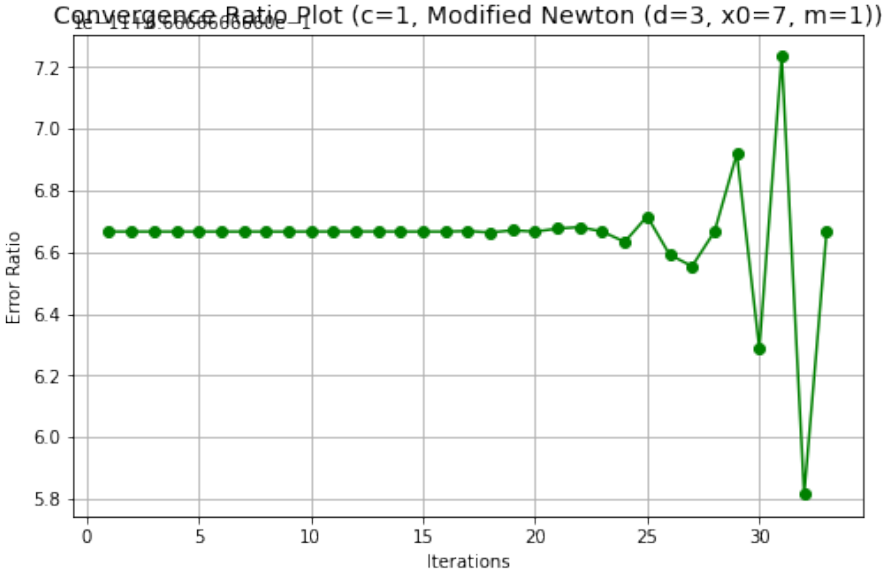




Parameter	Value
Initial Guess	7.0
Multiplicity (m)	1.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

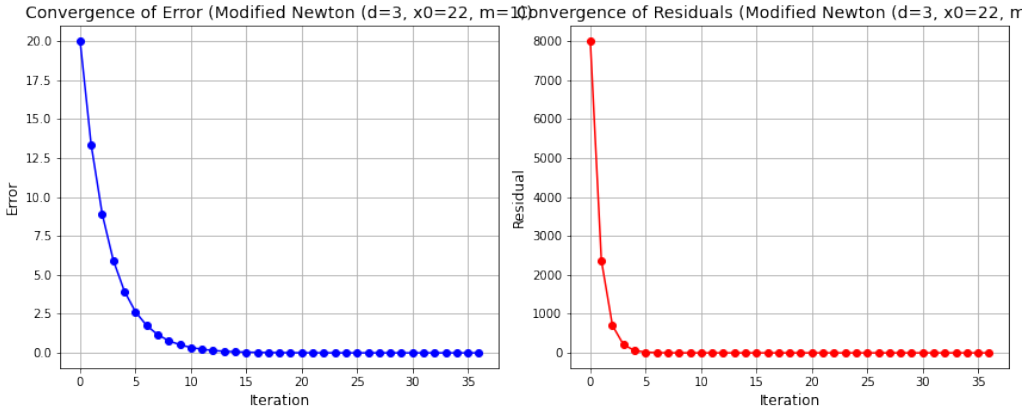
Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	7	2	1	7.73e-06	4.61e-16	33

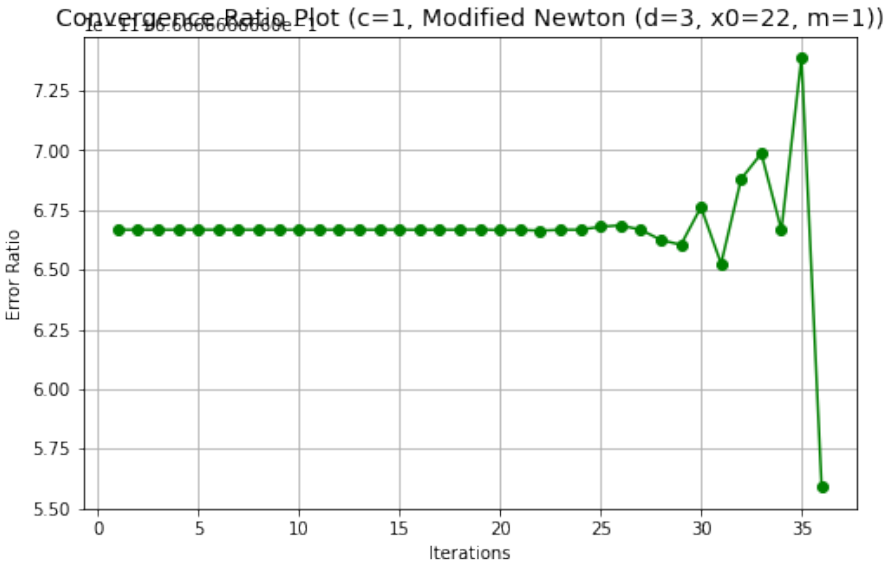




Parameter	Value
Initial Guess	22.0
Multiplicity (m)	1.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	22	2	1	9.16e-06	7.68e-16	36



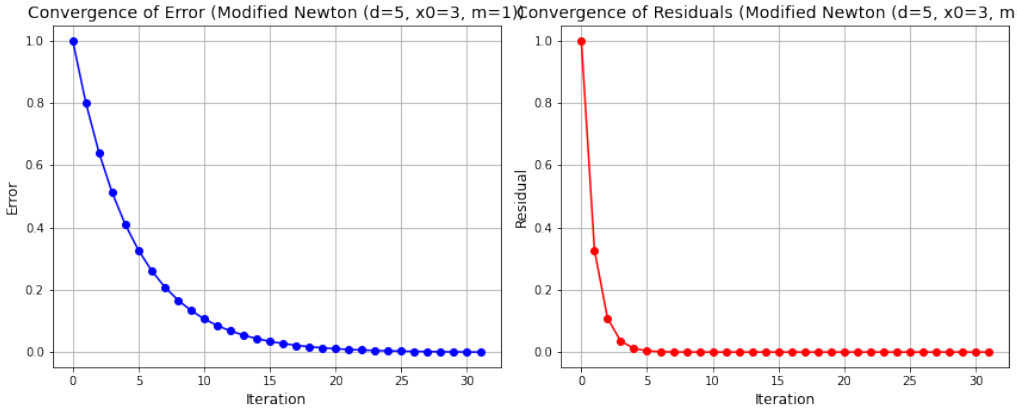


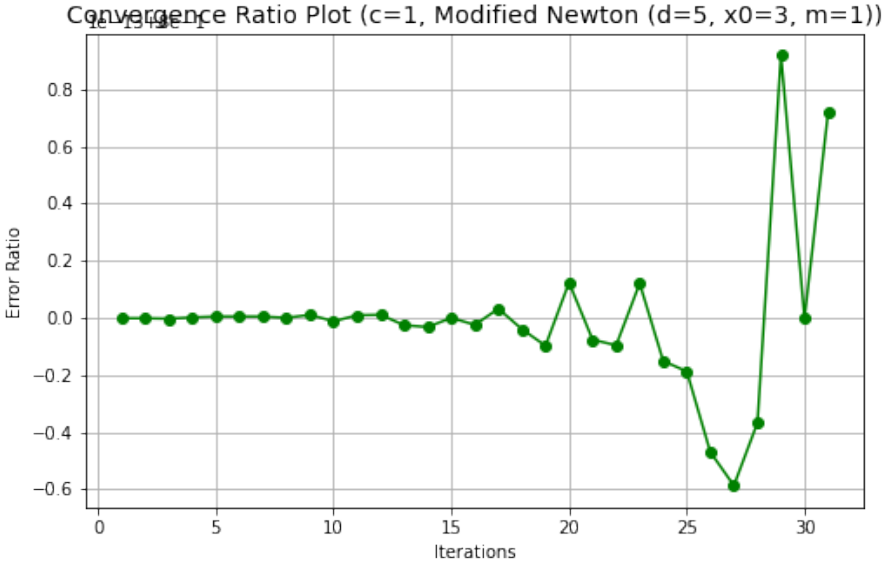
Conclusion: We get linear convergence in these cases, and the methods become slower (more iterations) as our guesses move away from the root. This behavior is unchanged. Uniquely, for higher degrees, we do get a more erratic behavior in convergence asymptotically, and it converges super-linearly in certain cases.

Let's now use much higher values of degree (d=5):

Parameter	Value
Initial Guess	3.0
Multiplicity (m)	1.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

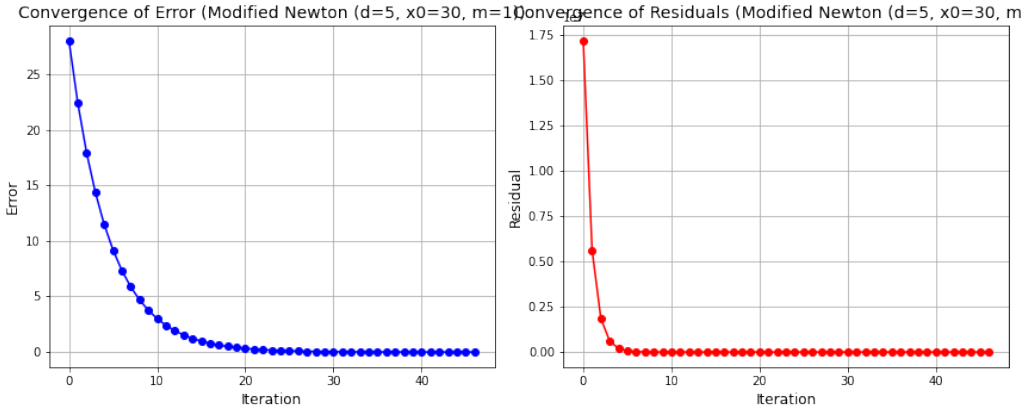
Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	3	2	1	9.90e-04	9.53e-16	31

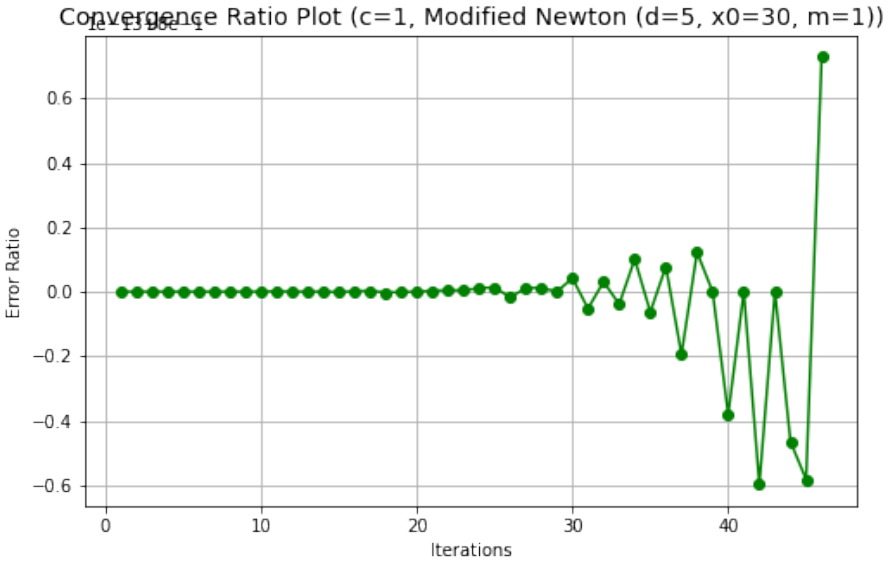




Parameter	Value
Initial Guess	30.0
Multiplicity (m)	1.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	30	2	1	9.76e-04	8.84e-16	46

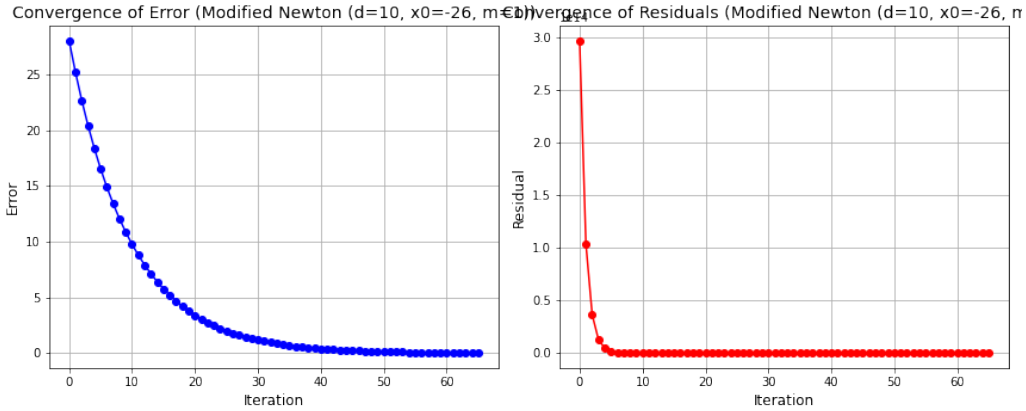


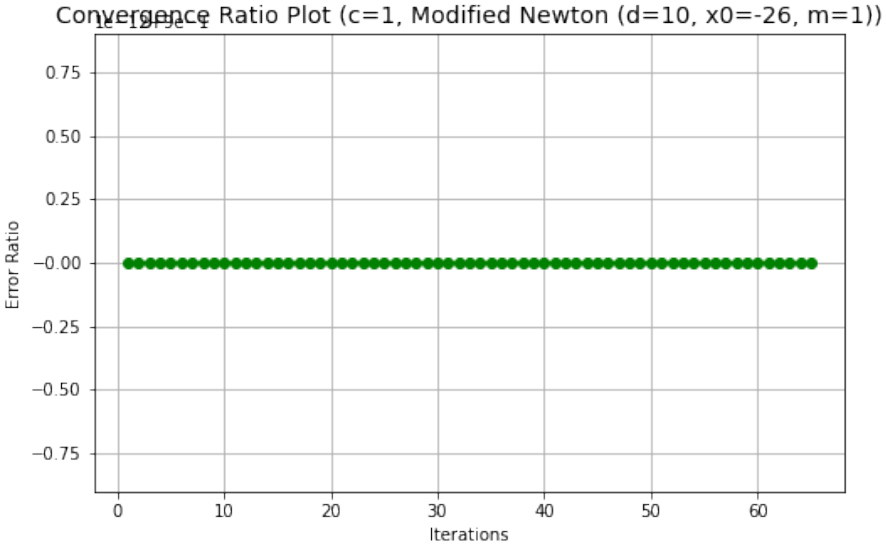


In the following we try an initial guess on the other side of the root, and use a much higher degree (d=10):

Parameter	Value
Initial Guess	-26.0
Multiplicity (m)	1.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	-26	2	1	2.97e-02	5.36e-16	65





Conclusion: It becomes obvious that in these cases it is increasingly hard to get any better than linear convergence (I suspect that erratic behavior of error ratios has to do with numerical pitfalls than any underlying effects).

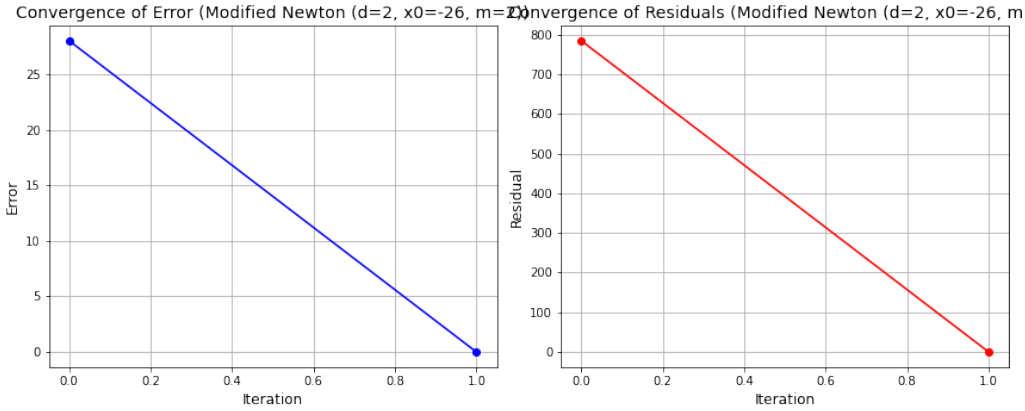
0.2 Using Modified Newton's Method

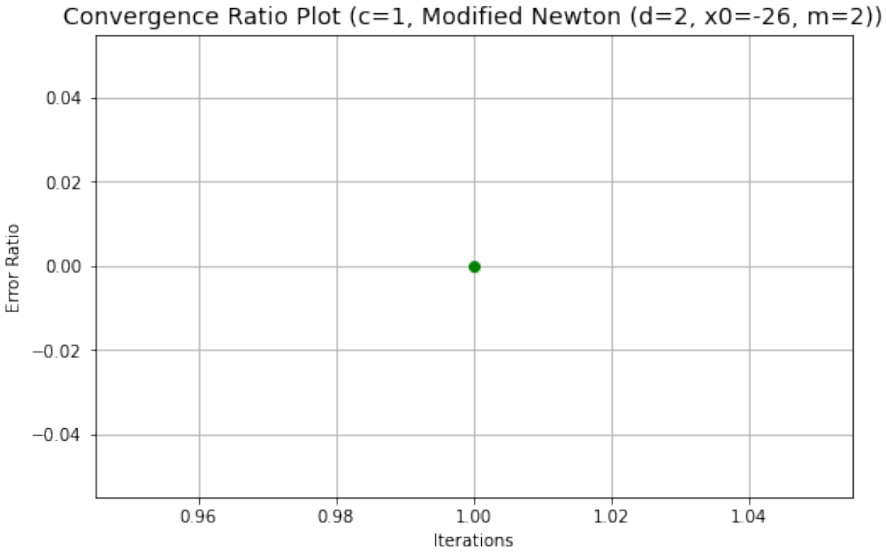
Use modified Newton's method with $m = d$, and empirically demonstrate the convergence rate trends as you take $d = 2, 3, \ldots$. You should demonstrate the behavior with two to three different x_0 for each d and record your observations on the behavior.

We begin by forcing d=m and running the same experiments as above:

Parameter	Value
Initial Guess	-26.0
Multiplicity (m)	2.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	-26	2	2	0.00e+00	0.00e+00	1

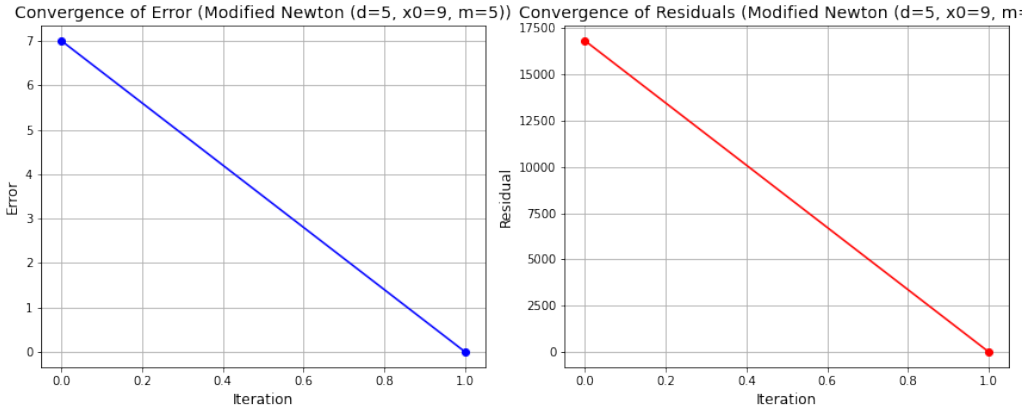


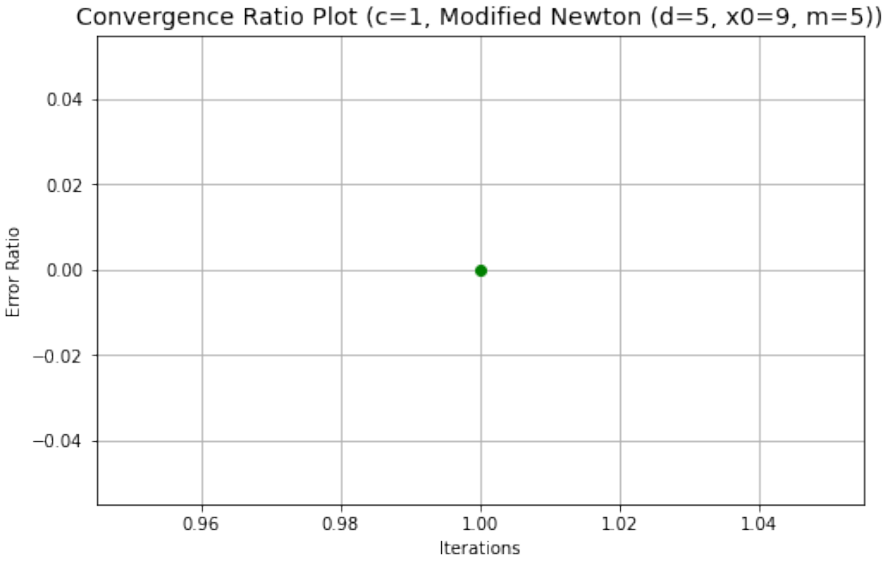


For different values of the initial guess, the results looked exactly the same as above, so I will not include those graphs for brevity on this point. We will still include graphs for other values of d=m. It turns out the plots look identical regardless!

Parameter	Value
Initial Guess	9.0
Multiplicity (m)	5.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	9	2	5	0.00e+00	0.00e+00	1





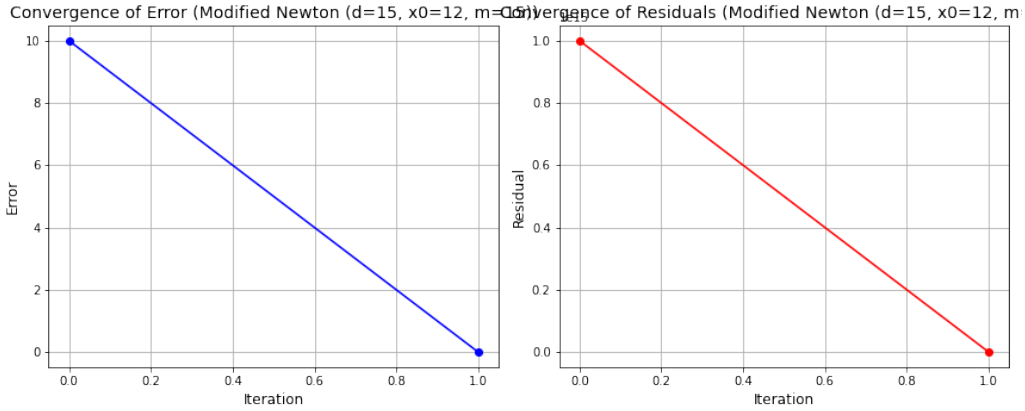
Do note that we can cause extremely slow algorithms by setting very large values of d and the initial guess. For example the following setup did not lead to a runtime 'timed-out' error:

```
# Experiment parameters
d = 15  # Multiplicity of the root
x_0 = 2 + 116  # Initial guess
r = 2  # Real root
TOL_error = 1e-8  # Error tolerance
TOL_residual = 1e-8  # Residual tolerance
MAX_ITER = 1000  # Maximum iterations
c = 1  # Convergence constant for the ratio plot
m = d  # Multiplicity parameter (m=1 corresponds to standard Newton's method)

# Run a single experiment
run_modified_newton_experiment(d=d, x_0=x_0, r=r, TOL_error=TOL_error, TOL_residual=TOL_residual, MAX_ITER=MAX_ITER, c=c, m=m)

1m 10.0s
```

For smaller values of the initial guess, we get the same results as above (convergence in 1 step).

Parameter	Value
Initial Guess	12.0
Multiplicity (m)	15.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	12	2	15	0.00e+00	0.00e+00	1



Conclusion: If d=m, we get convergence in 1 step, barring numerical pitfalls.

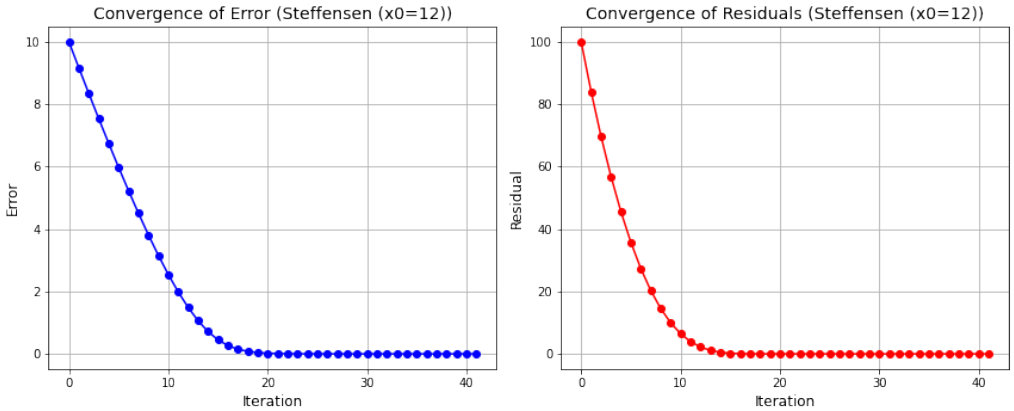
0.3 Steffensen's Method

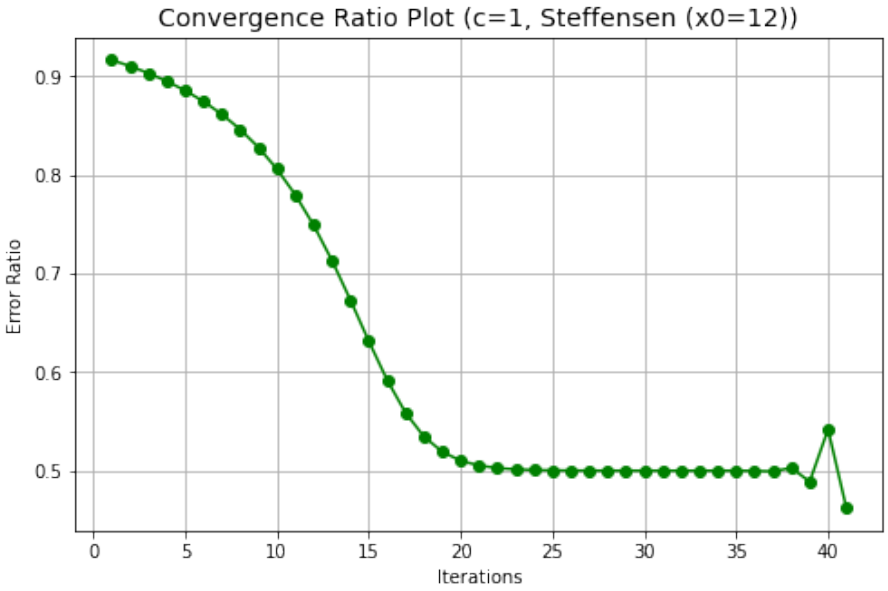
How fast does Steffensen's method converge for $d = 2, 3, \ldots$? Does it appear quadratic?

Consider the following experiments (next page): We have the following values for d = 2, 3, 4, and 8.

Parameter	Value
Initial Guess	12.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

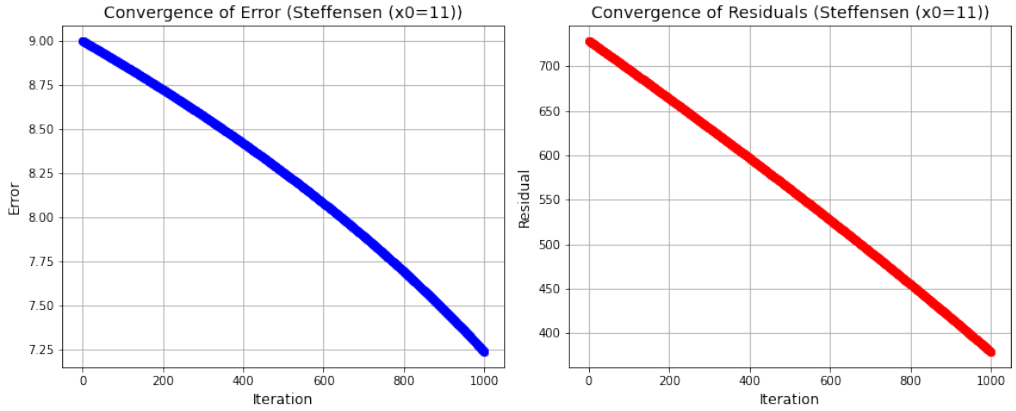
Method Used	Initial Guess	True Root	Final Error	Final Residual	Number of Iterations
Steffensen	12	2	1.01e-08	1.02e-16	41

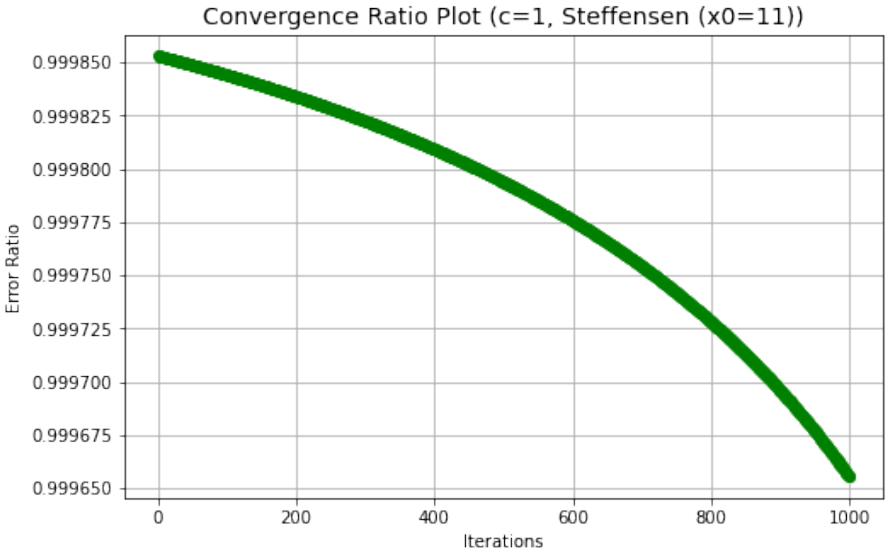




Parameter	Value
Initial Guess	11.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Final Error	Final Residual	Number of Iterations
Steffensen	11	2	7.23e+00	3.78e+02	1000

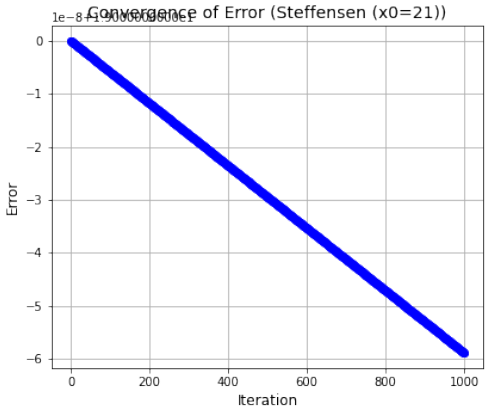


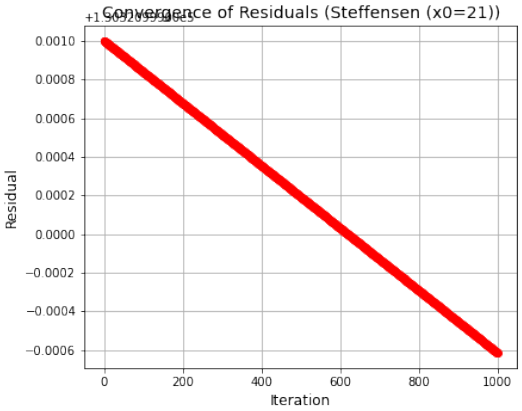


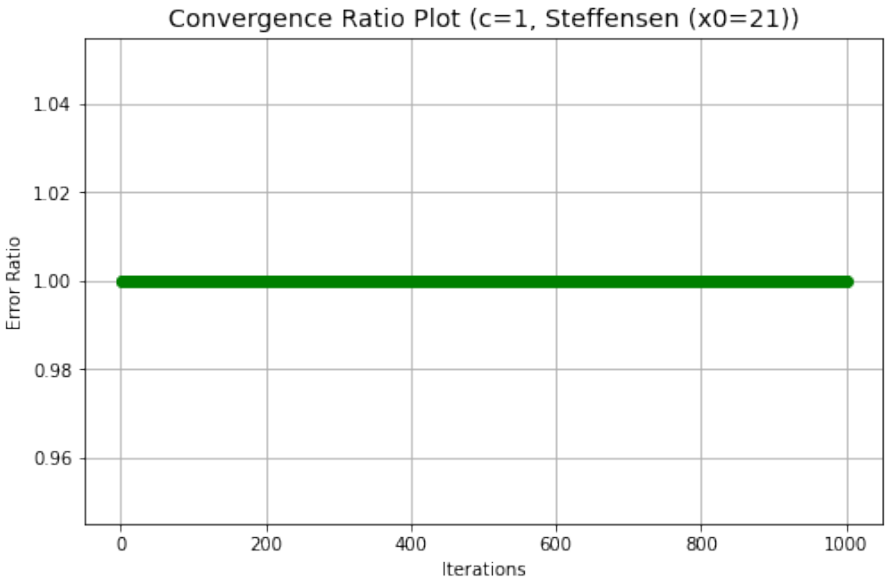
In these cases we get some super-linear convergence. For higher values of d, we see that Steffensen's does not seem to converge, unless we begin very close to the root:

Parameter	Value
Initial Guess	21.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Final Error	Final Residual	Number of Iterations
Steffensen	21	2	1.90e+01	1.30e+05	1000

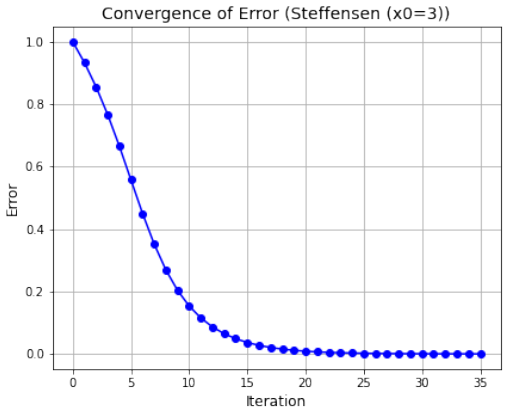


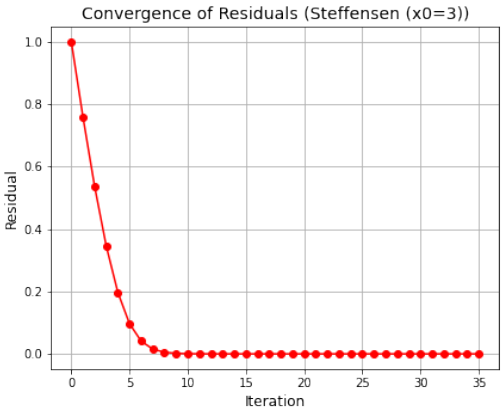


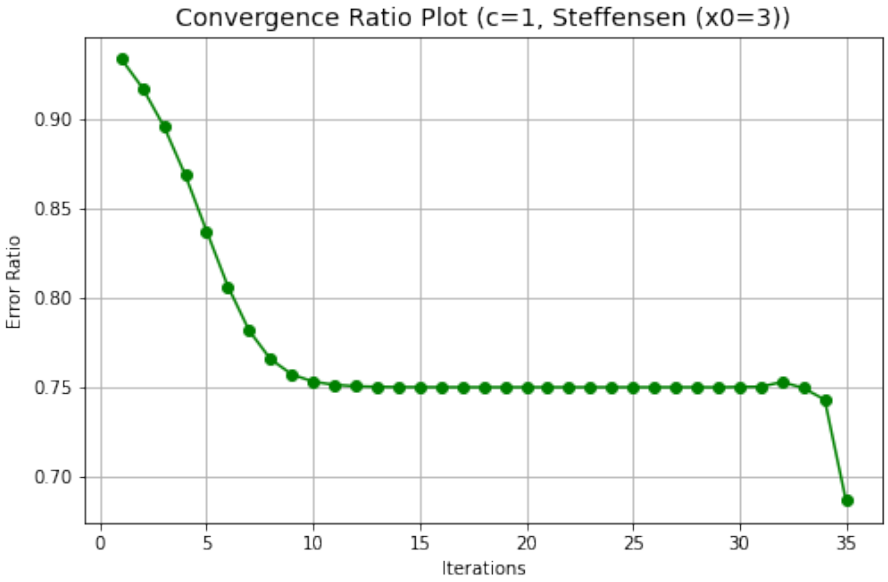


Parameter	Value
Initial Guess	3.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Final Error	Final Residual	Number of Iterations
Steffensen	3	2	1.05e-04	1.24e-16	35







Conclusion: Steffensen's converges faster than Newton (m=1) but is very sensitive to the initial guess, and further away from the root it can diverge. With higher and higher d, this sensitivity increases more and more. This makes sense since the difference approximation of the derivative becomes weaker the further away we are. It does appear to have quadratic convergence for when we start close to the root.

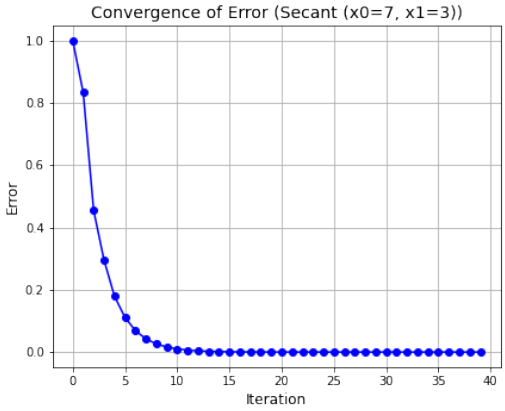
0.4 Behavior of Regula Falsi and Secant

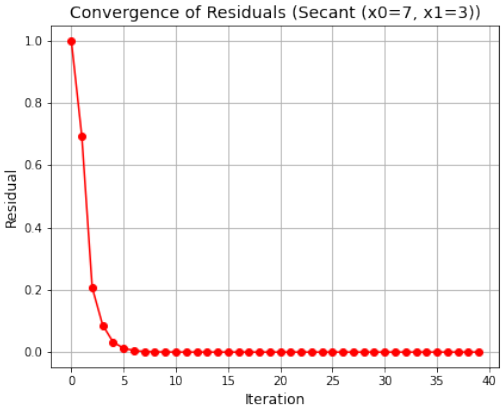
Describe and comment on the behavior of Regula Falsi and Secant for $d = 2, 3, \ldots$. How does your choice of the two initial points required for each affect the convergence behavior?

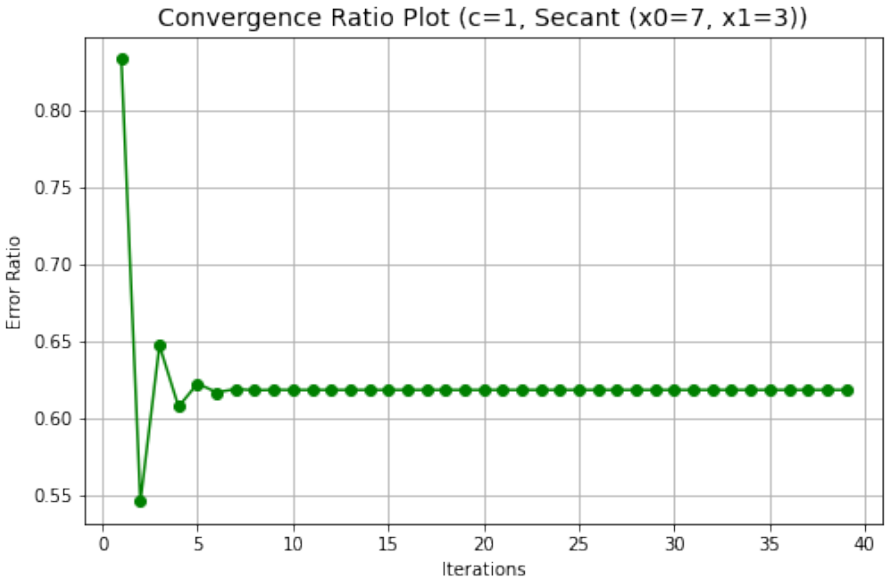
We first give the required results/plots for a few instances for both methods and then work with changing the initial points of the methods (the changes we can make include changing the distance between the points, changing the side of the root on which they lie etc.). Also note that Regula Falsi will only work for odd degree polynomials (in our implementation).

Parameter	Value
x_0	7.0
x_1	3.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	x_0	x_1	Final Error	Final Residual	Number of Iterations
Secant	7	2	7	3	8.70e-09	7.56e-17	39

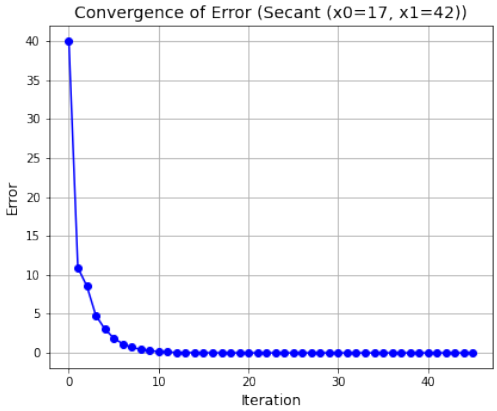


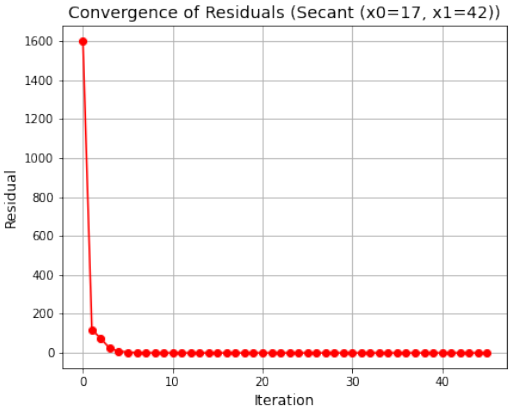


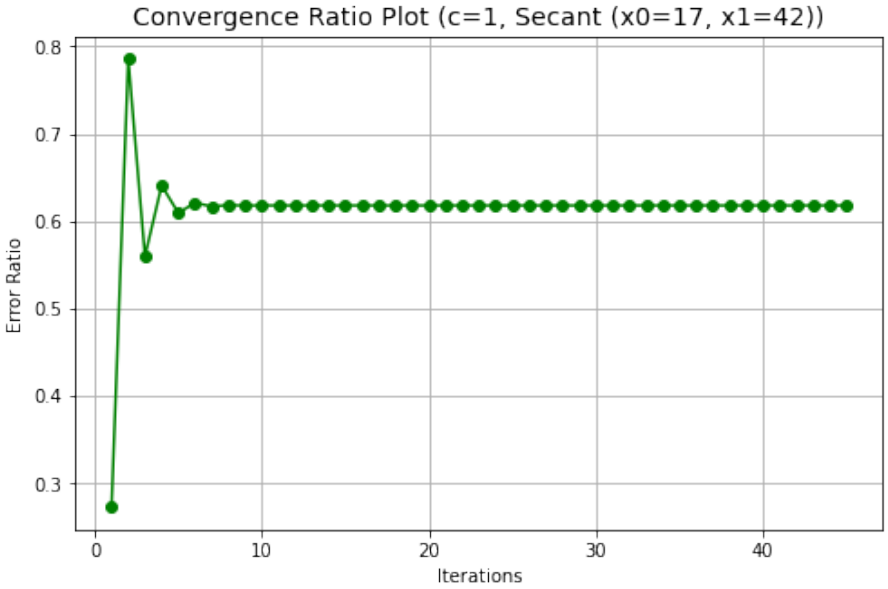


Parameter	Value
x_0	17.0
x_1	42.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	x_0	x_1	Final Error	Final Residual	Number of Iterations
Secant	17	2	17	42	8.23e-09	6.77e-17	45

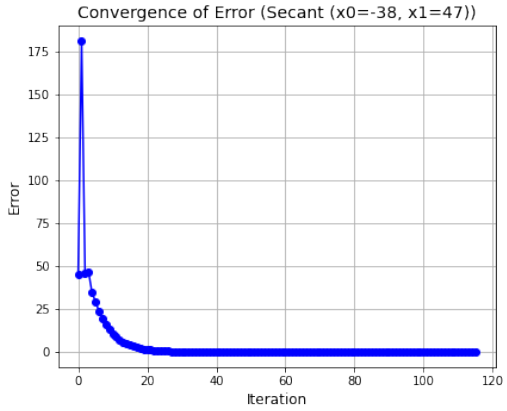


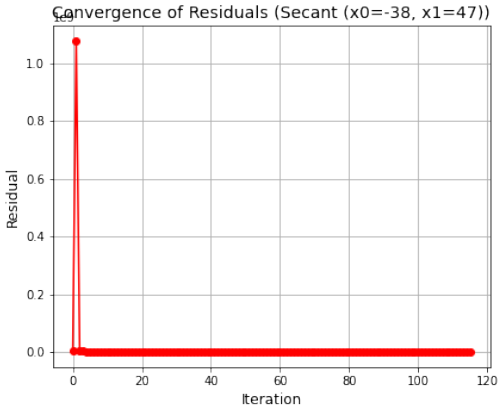


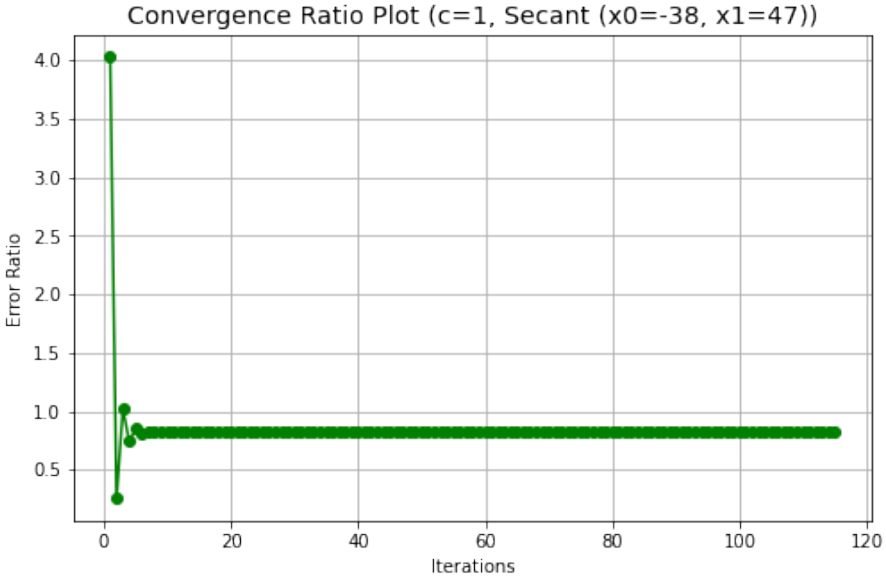


Parameter	Value
x_0	-38.0
x_1	47.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	x_0	x_1	Final Error	Final Residual	Number of Iterations
Secant	-38	2	-38	47	8.64e-09	5.57e-33	115

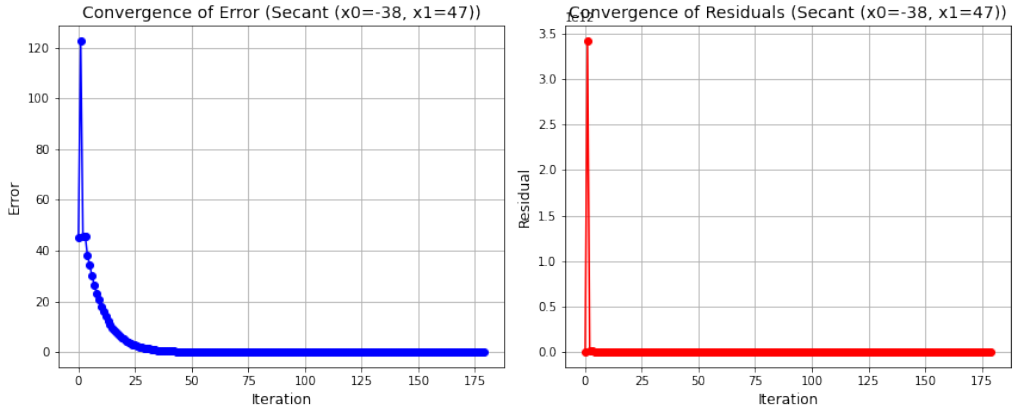


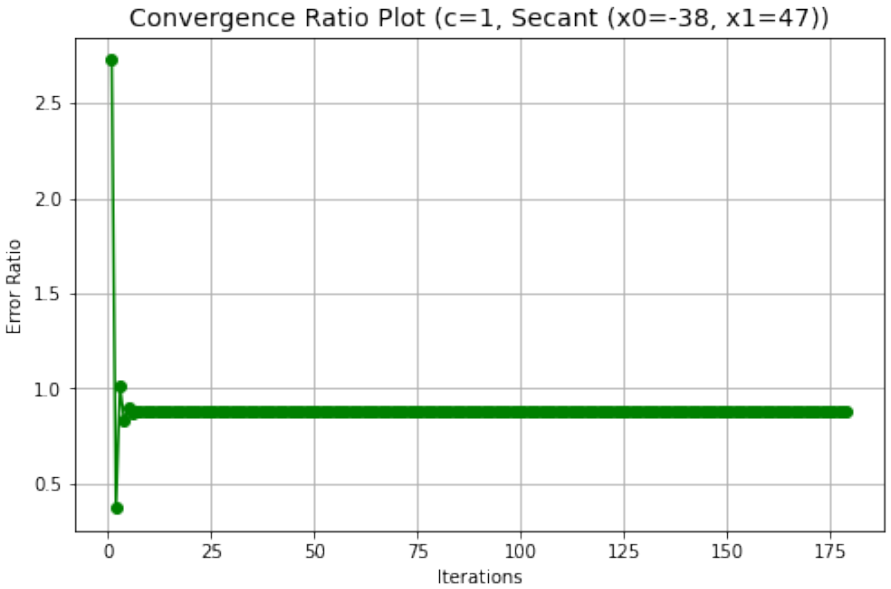




Parameter	Value
x_0	-38.0
x_1	47.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	x_0	x_1	Final Error	Final Residual	Number of Iterations
Secant	-38	2	-38	47	9.57e-09	7.69e-49	179

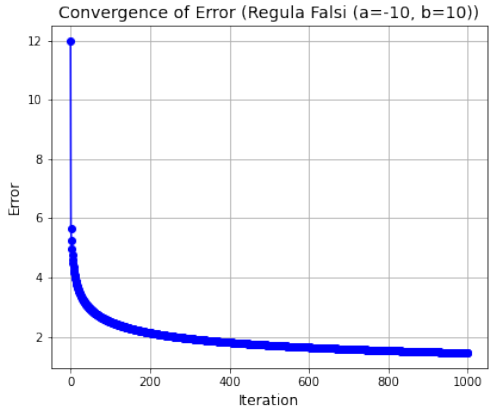


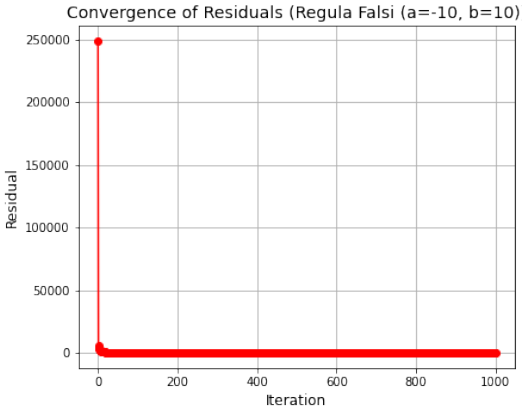


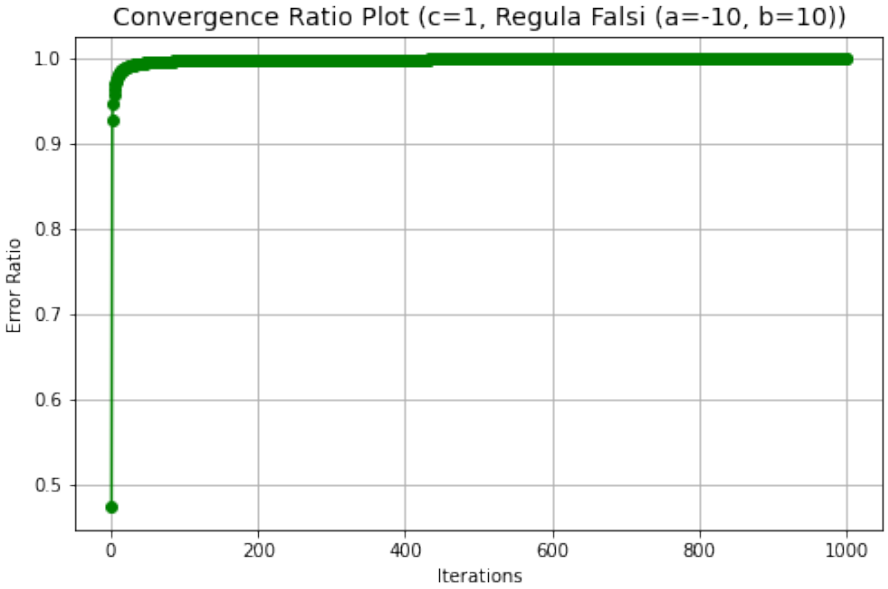
For Regula Falsi, we get (d=5):

Parameter	Value
a	-10.0
b	10.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	a	b	Final Error	Final Residual	Number of Iterations
Regula Falsi	-10	2	-10	10	1.45e+00	6.46e+00	1000



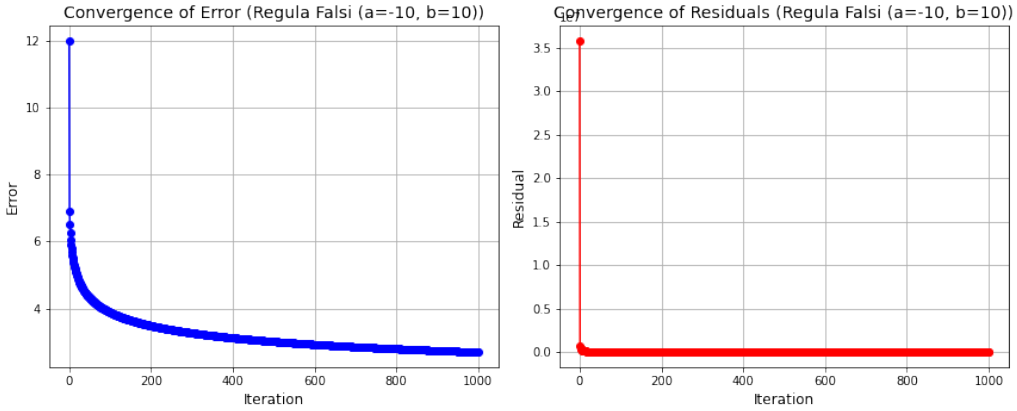


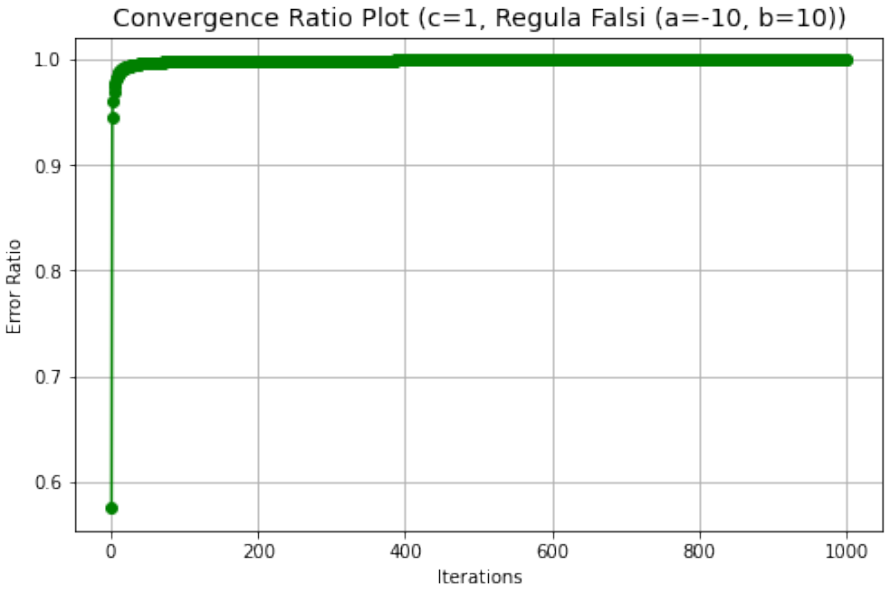


And now for d=7:

Parameter	Value
a	-10.0
b	10.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	a	b	Final Error	Final Residual	Number of Iterations
Regula Falsi	-10	2	-10	10	2.70e+00	1.06e+03	1000





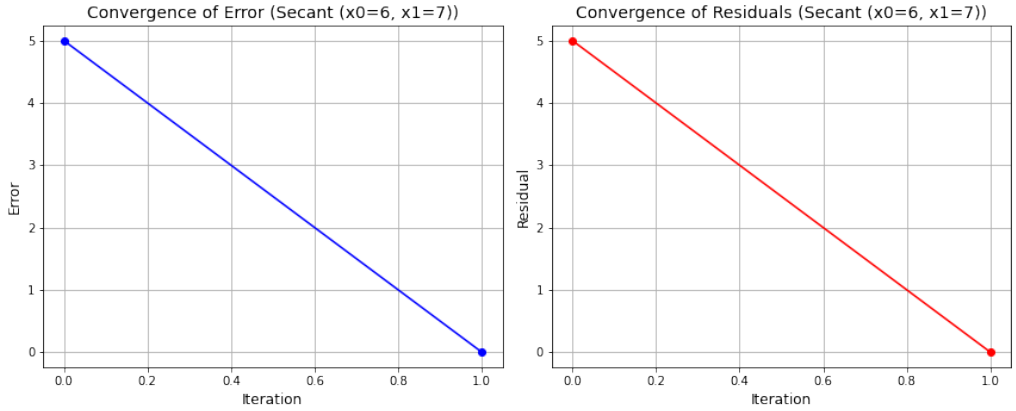
Conclusion: For the Secant Method, changing the overall distance of the two points from the root only makes the algorithm slower, and the convergence stays linear as in the control experiment. Increasing d only makes the algorithm slower, but the rate of convergence is pretty similar. More plots and results can be accessed from the folder submitted as part of this assignment, confirming these hypothesis. Similar results hold for Regula Falsi. Here the distance between the two points used has an even larger effect on the speed and convergence, but there is much less of an effect of degree on the speed. For even degrees, Regula Falsi does not proceed at all. It also has the worst convergence rate for when it does proceed. Finally, Regular Falsi also tends to misbehave if the interval we start with is too small! To achieve the same accuracy, we need a larger number of maximum allowed iterations.Thus our results hold only for odd degrees for Regula Falsi.

0.5 Special Case: $d = 1$

What happens for these methods when $d = 1$? We will show representative results (extra experiments are present in the repository). For Secant:

Parameter	Value
x_0	6.0
x_1	7.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

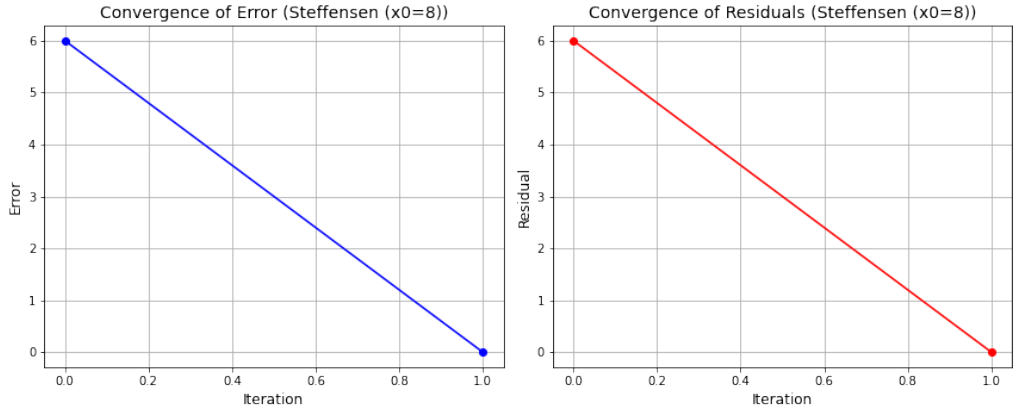
Method Used	Initial Guess	True Root	x_0	x_1	Final Error	Final Residual	Number of Iterations
Secant	6	2	6	7	0.00e+00	0.00e+00	1



For Steffensen's:

Parameter	Value
Initial Guess	8.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

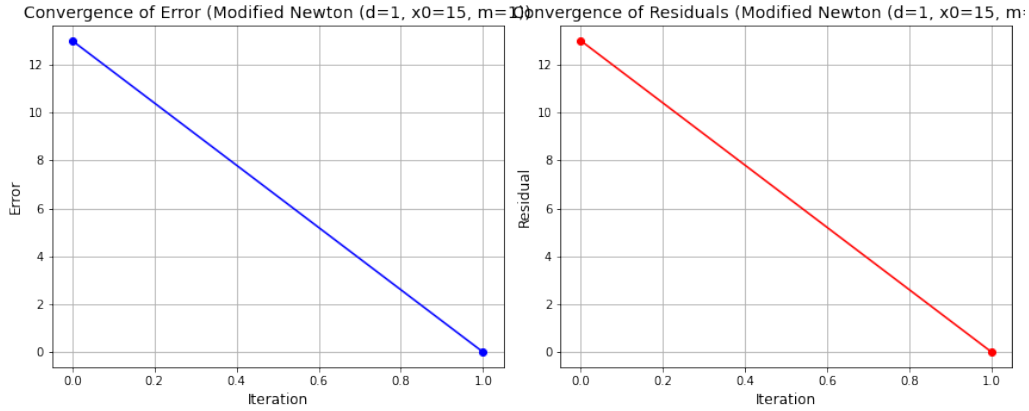
Method Used	Initial Guess	True Root	Final Error	Final Residual	Number of Iterations
Steffensen	8	2	0.00e+00	0.00e+00	1



Standard Newton's:

Parameter	Value
Initial Guess	15.0
Multiplicity (m)	1.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

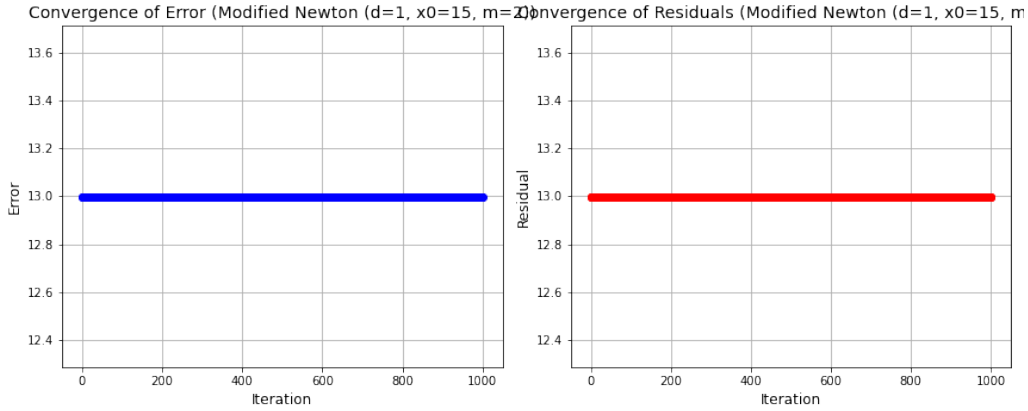
Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	15	2	1	0.00e+00	0.00e+00	1

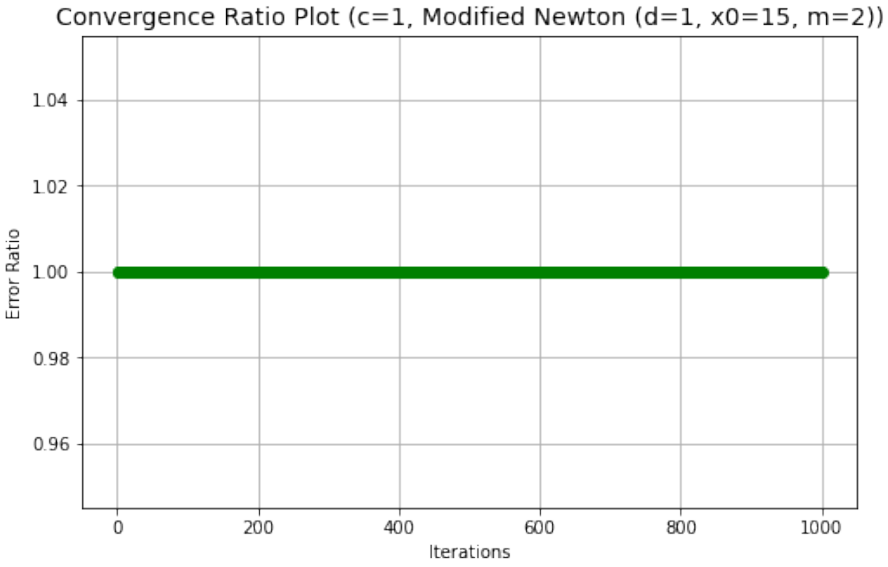


Finally, modified Newton (with m=2):

Parameter	Value
Initial Guess	15.0
Multiplicity (m)	2.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	15	2	2	1.30e+01	1.30e+01	1000





Conclusion: For reasonable choices of the starting parameters, most of the above algorithms (excluding Modified Newton and Regula Falsi) converge immediately (I skipped the error ratio plots since this is singular point). Regula Falsi in fact also converges immediately but the algorithm does not proceed. This can be explained - the line connecting f(a) and f(b) passes through the root r by definition, which Regula Falsi jumps to at iteration 0 by design. Modified Newton does not converge if d=1 and $m > 1$. This will be tested more thoroughly in the next section.

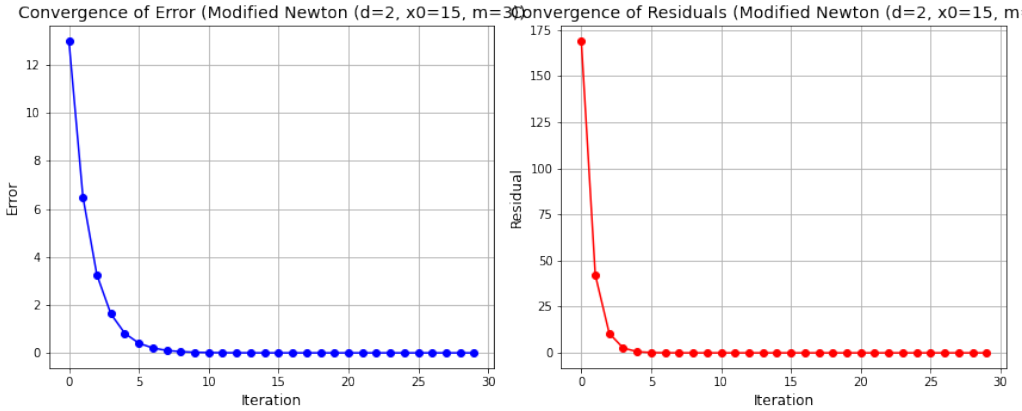
0.6 Modified Newton's Method with $m \neq d$

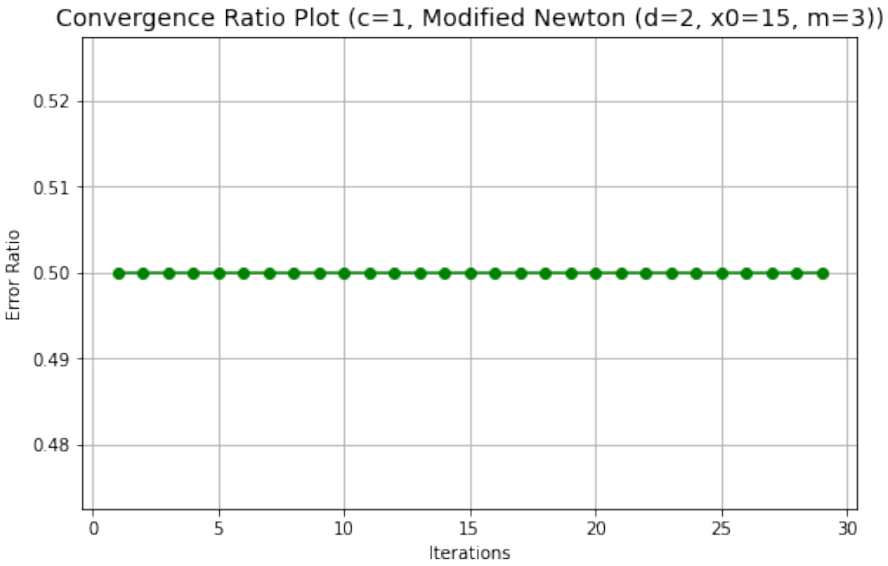
Suppose $d \geq 2$ and $m \neq d$, i.e., suppose you have set m to something other than the degree of the root ρ. What is the behavior of modified Newton's method for $m > d$ and $m < d$? Does it still converge? If so, what rate does it appear to have? Does it diverge for any of your choices of m, d, and x_0?

We restrict to $2 = d > m$ first since the other condition has already been tested when we tested the standard Newton and m=d.

Parameter	Value
Initial Guess	15.0
Multiplicity (m)	3.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	15	2	3	2.42e-08	5.86e-16	29

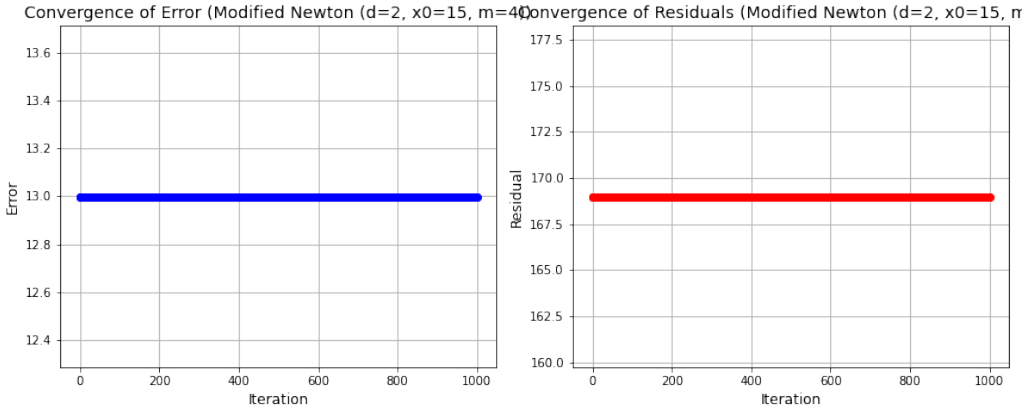


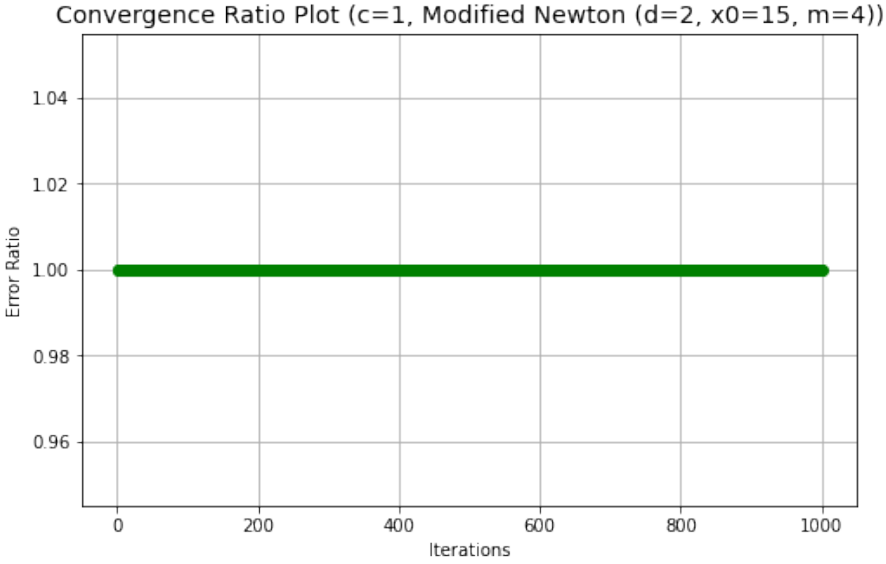


We clearly have linear convergence in this case. Lets try to increase the values of m. We immediately get no convergence:

Parameter	Value
Initial Guess	15.0
Multiplicity (m)	4.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

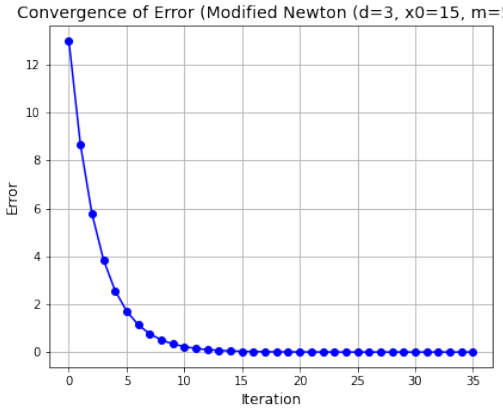
Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	15	2	4	1.30e+01	1.69e+02	1000

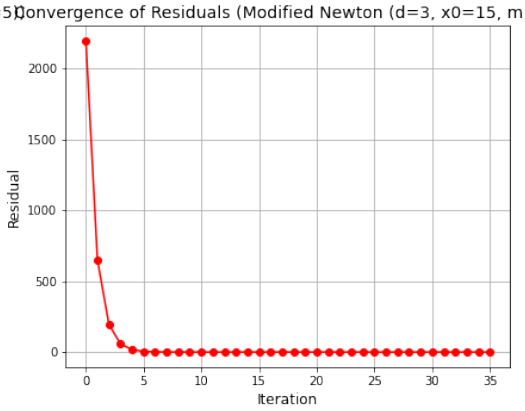


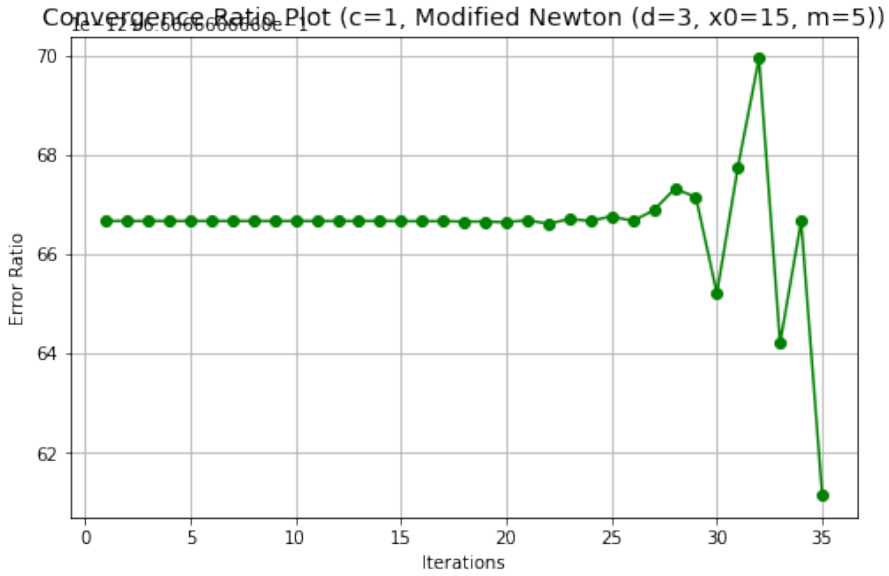


Parameter	Value
Initial Guess	15.0
Multiplicity (m)	5.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	15	2	5	8.93e-06	7.12e-16	35

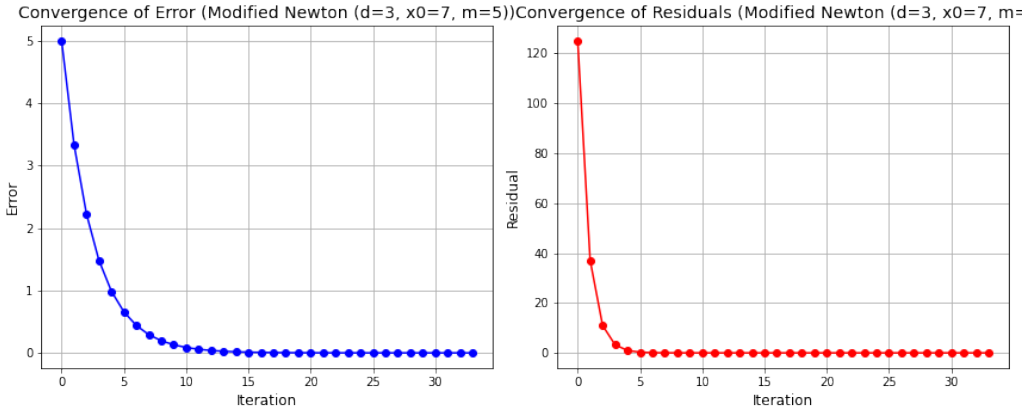


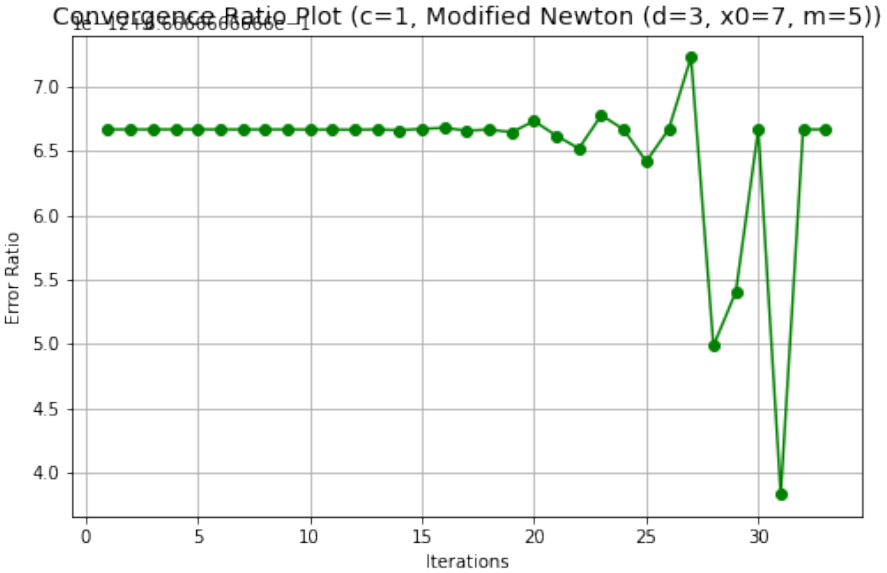




Parameter	Value
Initial Guess	7.0
Multiplicity (m)	5.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

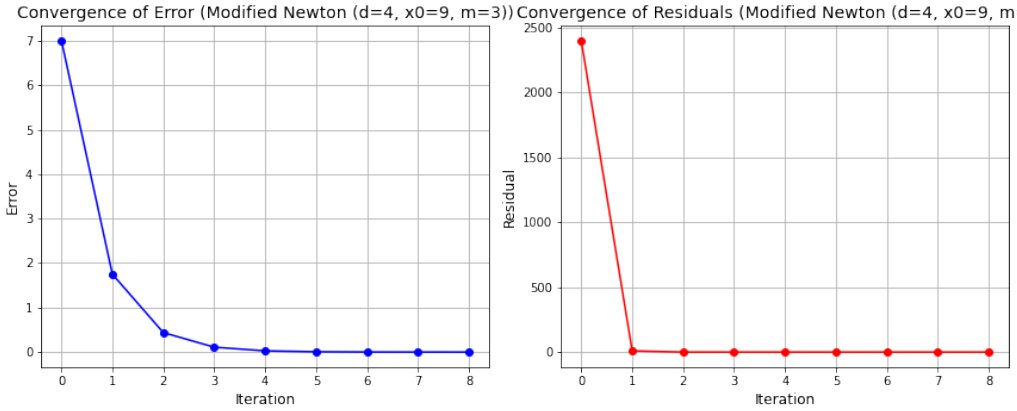
Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	7	2	5	7.73e-06	4.61e-16	33

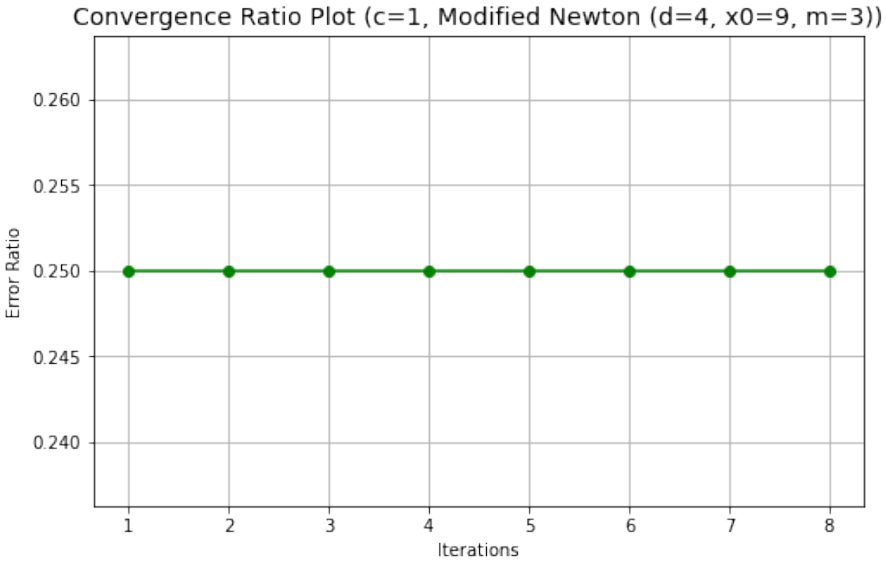


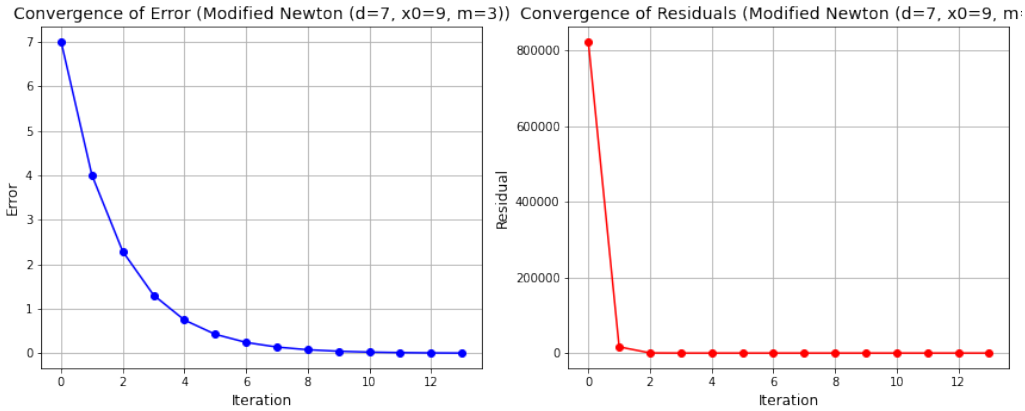


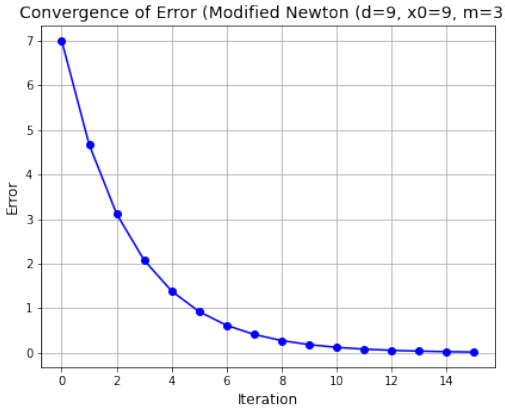
Parameter	Value
Initial Guess	9.0
Multiplicity (m)	3.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

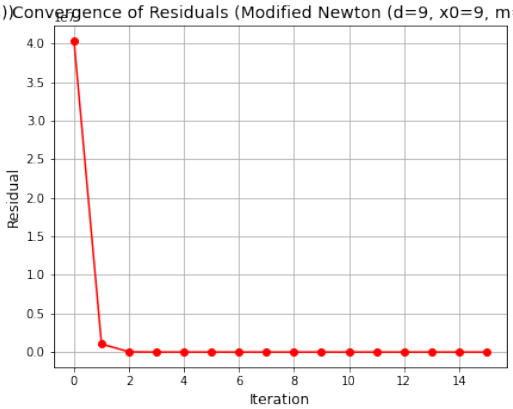
Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	9	2	3	1.07e-04	1.30e-16	8




Convergence Ratio Plot (c=1, Modified Newton (d=4, x0=9, m=3))
Error Ratio
0.260
0.255
0.250
0.245
0.240
1
2
3
4
5
6
7
8
Iterations


Convergence of Error (Modified Newton (d=7, x0=9, m=3))
Error
Iteration
Convergence of Residuals (Modified Newton (d=7, x0=9, m
Residual
800000
600000
400000
200000
0
Iteration





Parameter	Value
Initial Guess	9.0
Multiplicity (m)	3.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Parameter	Value
Initial Guess	9.0
Multiplicity (m)	3.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	9	2	3	4.85e-03	6.30e-17	13

Method Used	Initial Guess	True Root	Multiplicity (m)	Final Error	Final Residual	Number of Iterations
Modified Newton	9	2	3	1.60e-02	6.82e-17	15

Conclusion: For m=5 and d=2, we actually immediately get numerical overflow errors and the algorithm cannot proceed. This was the case for arbitrary initial guesses of the root, even ones that were very close. I suspect this extrapolates as a general fact i.e. if $d << m$, then we may run into numerical issues, particularly overflow. This was supported by the fact that we recovered convergence (albeit linear) if we now turn d=3 and keep m=5. In these cases, we get convergence profiles similar to the standard newton algorithm for higher multiplicity roots! The smaller the distance of $d < m$ the faster the algorithm tends to go (e.g. 7,8, see repository). If $m < d$, we also get results matching the standard algorithm, which degrade as d goes up.

1 Distinct Roots and Trends as They Coalesce: 3 Roots and Newton Method Comparison with Other Methods

Consider the generic cubic polynomial used in the problem with three distinct roots 0, $\rho > 0$ and $-\rho$ with the following form and properties:

$$f(x) = x(x-\rho)(x+\rho) = x^3 - \rho^2 x, \quad \rho > 0$$

$$f'(x) = 3x^2 - \rho^2 = 3(x^2 - \alpha^2) = 3(x-\alpha)(x+\alpha), \quad \alpha = \frac{\rho}{\sqrt{3}} > 0$$

We previously analyzed the behavior of Newton's method on the intervals

$$x > \rho$$

$$\alpha < x < \rho$$

$$-\xi < x < \xi$$

where ξ is the "cycling point" of Newton's method for this $f(x)$.

1.1 Cycling Behavior and Floating-Point Considerations

When examining the cycling behavior, specifically address the observed behavior considering that ξ is almost certainly not representable exactly as a floating-point number, and therefore you are actually using an x_0 that is a floating-point number slightly different from the actual $\xi \in \mathbb{R}$. Consider the behavior as you take $x_0 = \xi \pm \epsilon$ for $\epsilon > 0$ with increasing values. Does the iteration actually cycle when $x_0 = \xi$ is used, i.e., is x_0 the floating-point version of ξ given when you evaluate it in floating-point?

1.2 Values of x_0 for Convergence to Roots

For what values of x_0 does Newton's iteration move far away from x_0 yet still converges to one of the roots?

1.3 Failure of Newton's Method in Floating-Point Systems

Is it possible to make Newton's method fail by producing numbers that are undefined or too large for the floating-point system?

1.4 Comparison with Other Methods

Compare the behavior of Regula Falsi, Secant, and Steffensen's methods to Newton's method. What choices of the initial two points for Regula Falsi and Secant cause trouble for the methods? Does Steffensen's method demonstrate quadratic convergence like Newton's method? If so, does it do so using the same x_0 values as Newton's method?

Newton's method, applied to the given cubic polynomial $f(x) = x(x-\rho)(x+\rho)$, exhibits cycling behavior due to floating-point representations, and specifically, the cycling point ξ, introduces slight perturbations when $x_0 = \xi + \epsilon$ is used. As $\epsilon > 0$ increases, the iteration may deviate significantly, especially if the computed trajectory lies close to the cycling point. For convergence, x_0 values far from ξ but within the attraction basin of a root can still lead to convergence.

However, floating-point errors can sometimes cause Newton's method to fail if the iterations produce extremely large or undefined values, especially when the derivative approaches zero. Comparing Newton's method to other methods: Regula Falsi, Secant, and Steffensen's, the initial choice of interval or points decides where convergence occurs: Regula Falsi and Secant require well-chosen intervals or initial guesses to avoid stalling, whereas Steffensen's method exhibits quadratic convergence similar to Newton's under ideal conditions. However, it may also fail in cases of poor initial guesses. The methods become highly sensitive to initial guesses in these conditions.

2 Distinct Roots and Trends as They Coalesce: Scaling

Now suppose you replace the function $f(x)$ with $\tilde{f}(x) = \sigma f(x)$ where $\sigma > 0$.

2.1 Effect on Newton's Method

How does this affect the behavior of Newton's method as σ grows larger?

Newton's method becomes slower as σ grows larger (experiments ran in the code). This makes sense: since derivatives are linear, the scaling cancels out in the update step. This means that the algorithm sequence $(x_k)_{k\geq 0}$ remains unchanged, but due to the scaling it takes longer for the algorithm to converge to the root. Another way to explain this is the the derivative is scaled with σ, meaning the slop grows, and increases the distance from the root on successive steps.

2.2 Effect on Other Methods

How does this affect the behavior of the other methods?

The performance of Steffensen's and Secant is affected similarly, but the performance of Regula Falsi degrades less due to scaling.

3 Distinct Roots and Trends as They Coalesce: Root Coalescing

Consider the function

$$\tilde{f}(x) = x(x-\rho_1)(x-\rho_2), \quad \rho_1 > 0, \quad \rho_2 > 0$$

that has roots $0 < \rho_1 < \rho_2$.

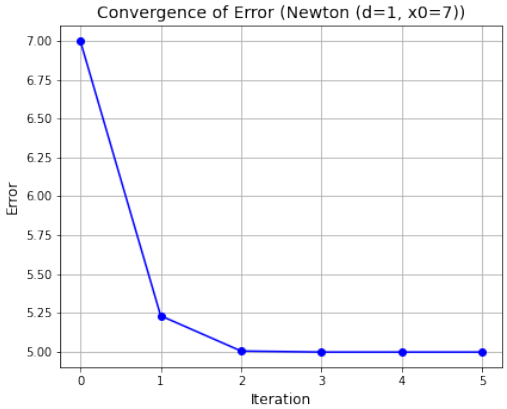
These questions investigate the gradual degradation of the convergence rate from quadratic to linear to slower linear, i.e., linear with a larger contraction constant (see the slides on this as a function of the degree of the root).

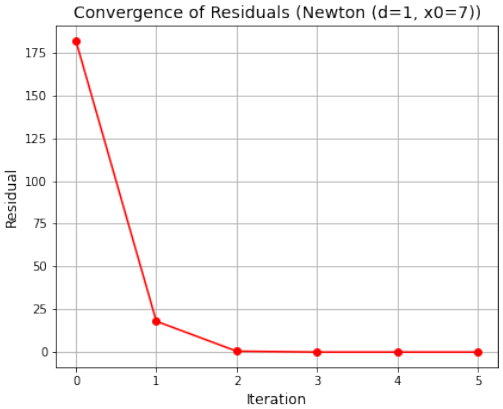
Note that for each such $\rho_1, \rho_2 > 0$, both of these values are roots of the function. Thus, the algorithm is susceptible to converge to either of the three roots (including 0). In the limit as these values approach zero we approximate the function $g(x) = x^3$ which has a root at 0 of multiplicity 3. This is where we expect to see the degradation. Note that if only one value approaches 0, the root at 0 now has multiplicity 2. We ignore the theoretical considerations in the succeeding discussion, and present the empirical results.

3.1 Effect of ρ_1 Approaching Zero

What happens to the behavior of Newton's method as you take ρ_1 closer and closer to 0?

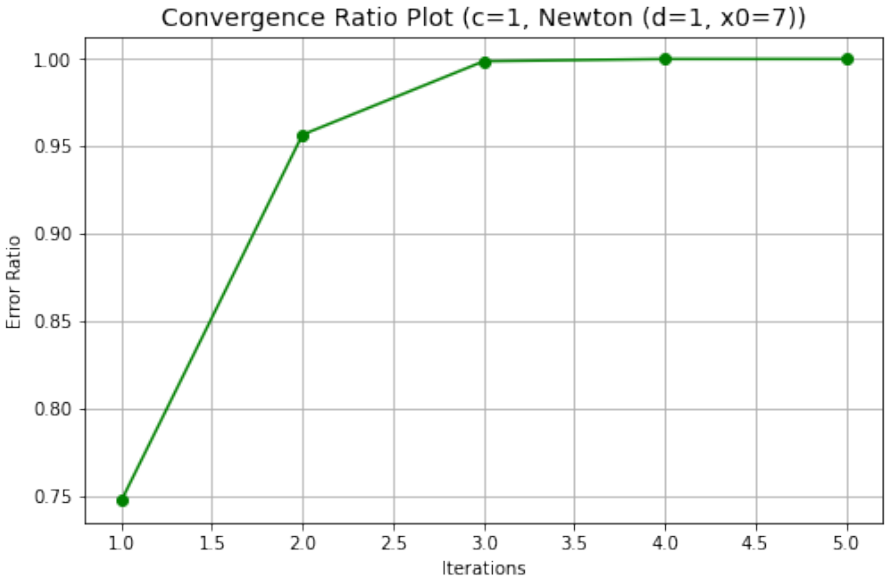
We fix $\rho_2 = 20$ or some other relatively larger value to preclude convergence to this root for our purposes. Since the other root will approach 0 from the right, we see that our root is more likely to converge to this value if our guess is bigger than ρ_1. For example:





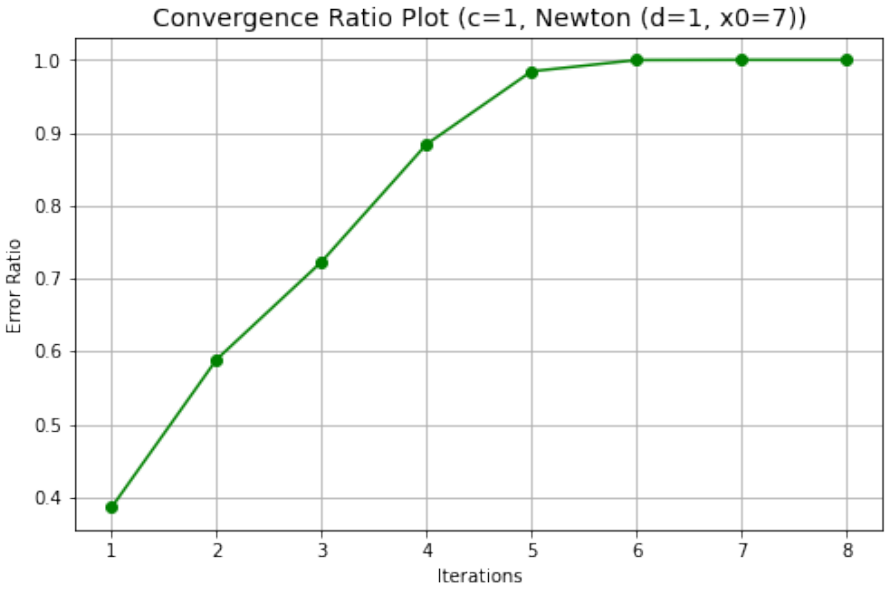
Parameter	Value
Initial Guess	7.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

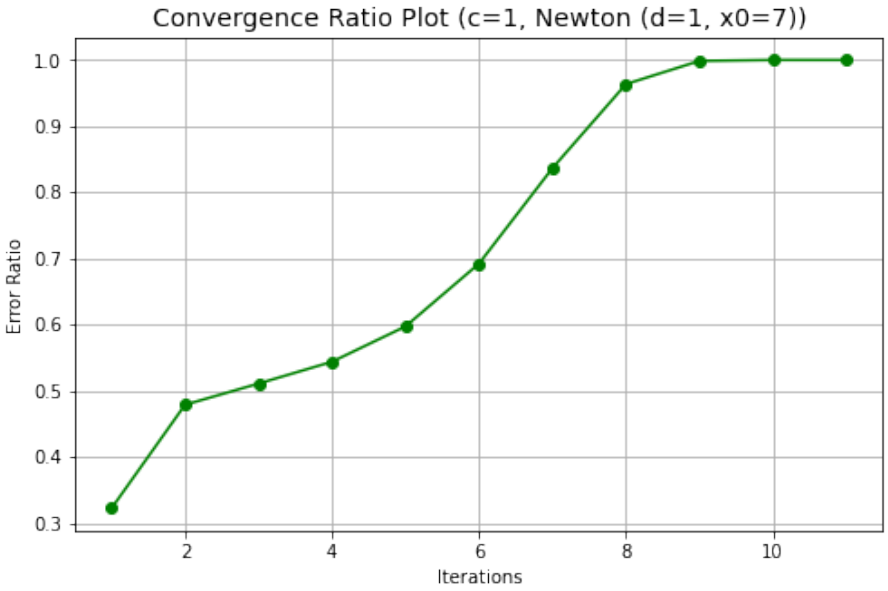
Method Used	Initial Guess	True Root	Final Error	Final Residual	Number of Iterations
Newton	7	0	5.00e+00	0.00e+00	5

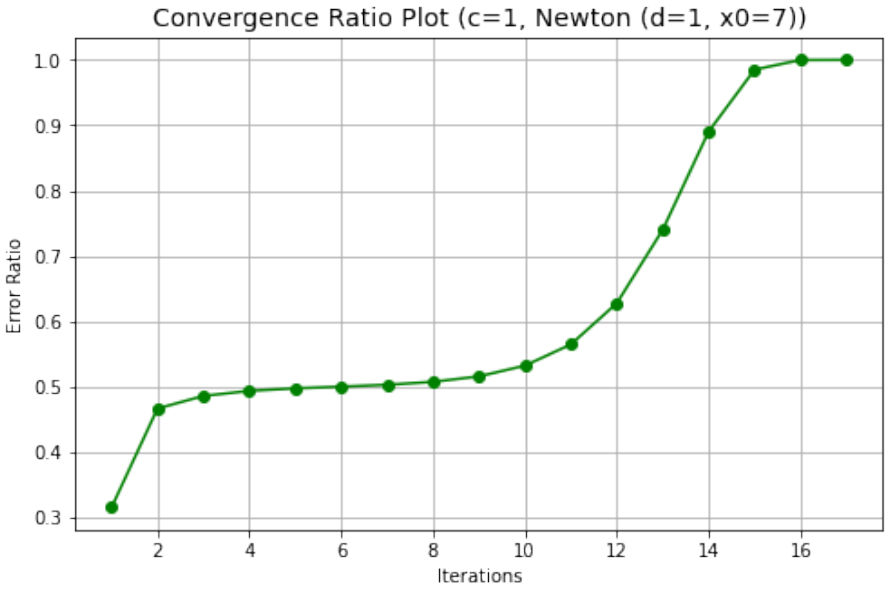


Clearly, we have linearly converged to ρ_1.

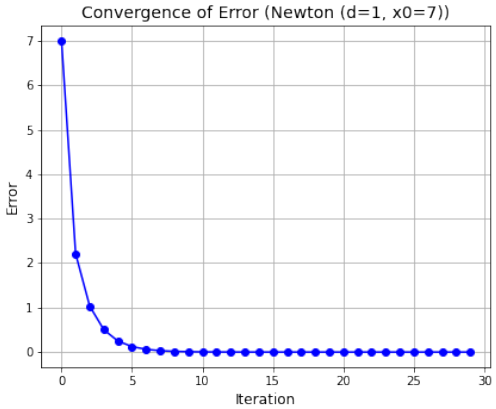
I will now show only the error ratio plots to demonstrate the degrading convergence profile.

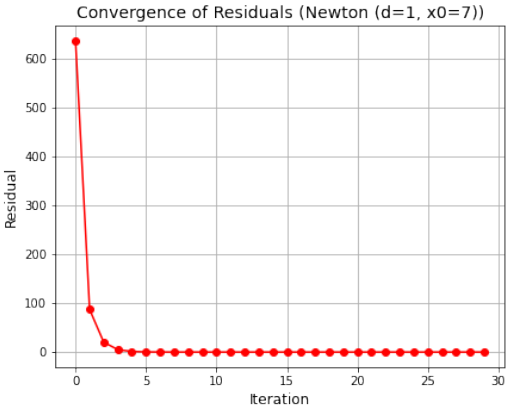






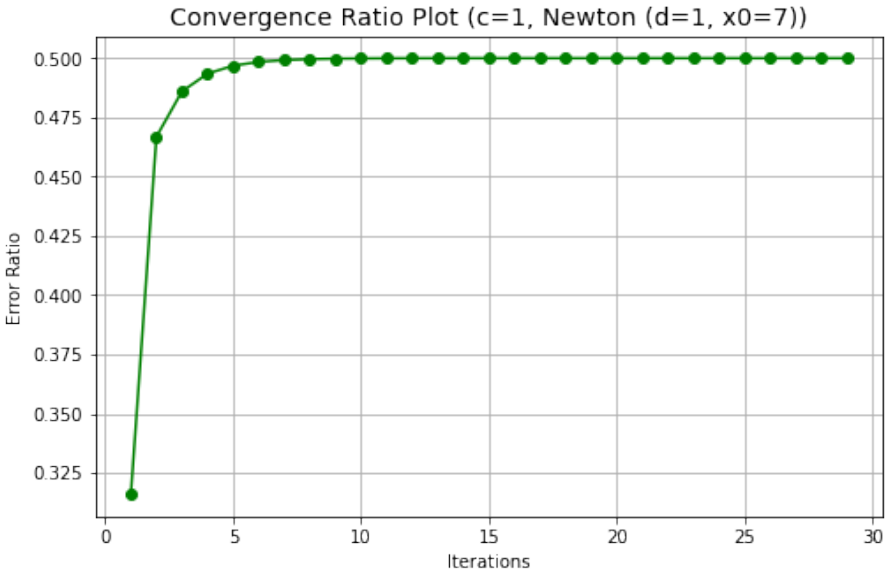
We can clearly see the convergence improve. In the limit we get





Parameter	Value
Initial Guess	7.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Final Error	Final Residual	Number of Iterations
Newton	7	0	7.29e-09	1.06e-15	29

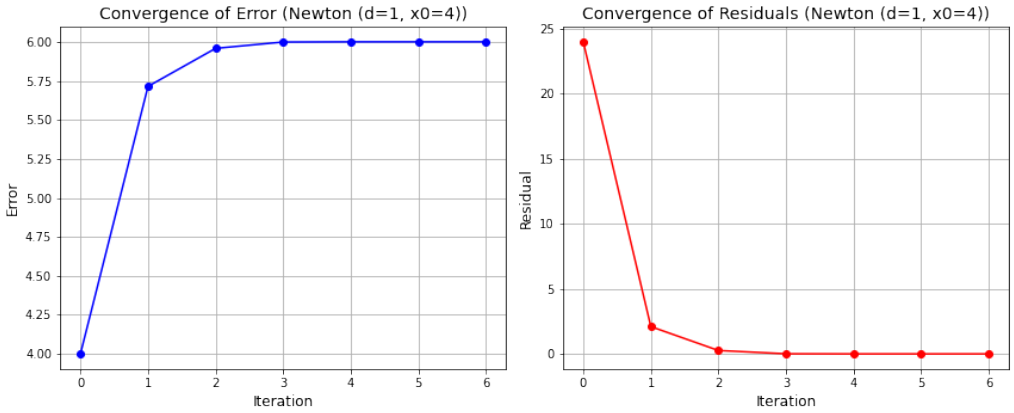


The convergence reverts back to what we started with since the root at 0 is now degenerate with multiplicity 2.

3.2 Effect of Both ρ_1 and ρ_2 Approaching Zero

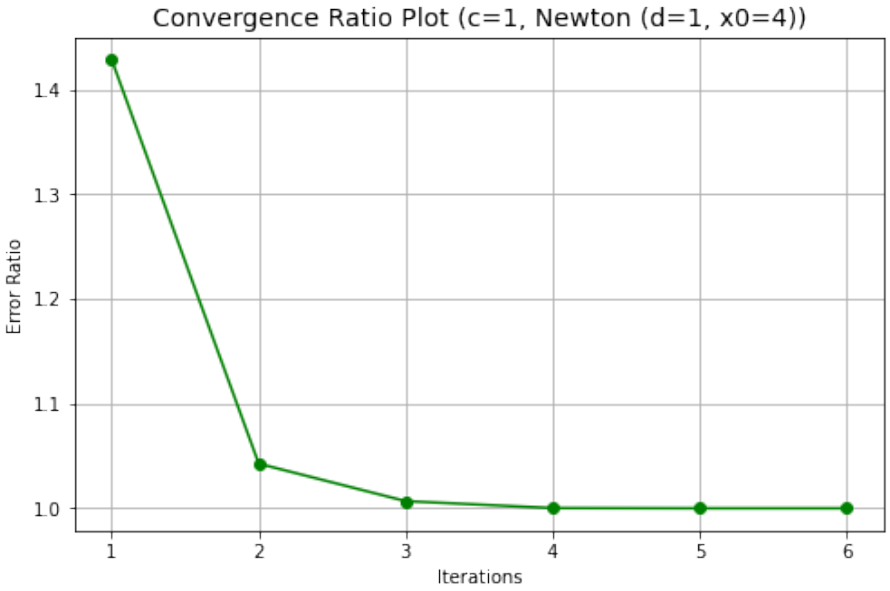
What happens to the behavior of Newton's method as you take ρ_1 and ρ_2 closer and closer to 0?

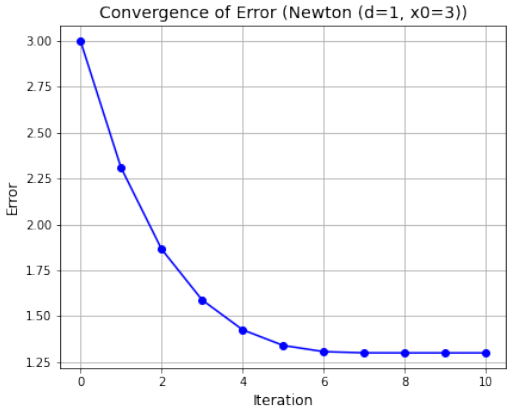
We show a series of experiments that model this behavior in succession:

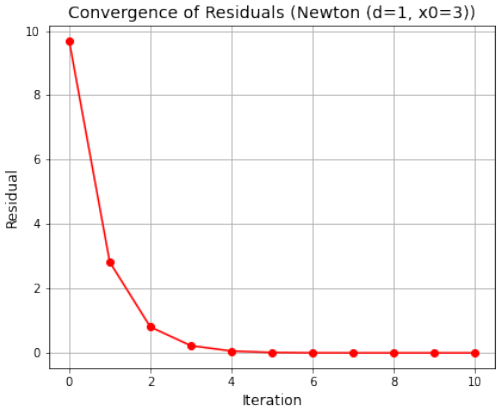


Parameter	Value
Initial Guess	4.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Final Error	Final Residual	Number of Iterations
Newton	4	0	6.00e+00	0.00e+00	6

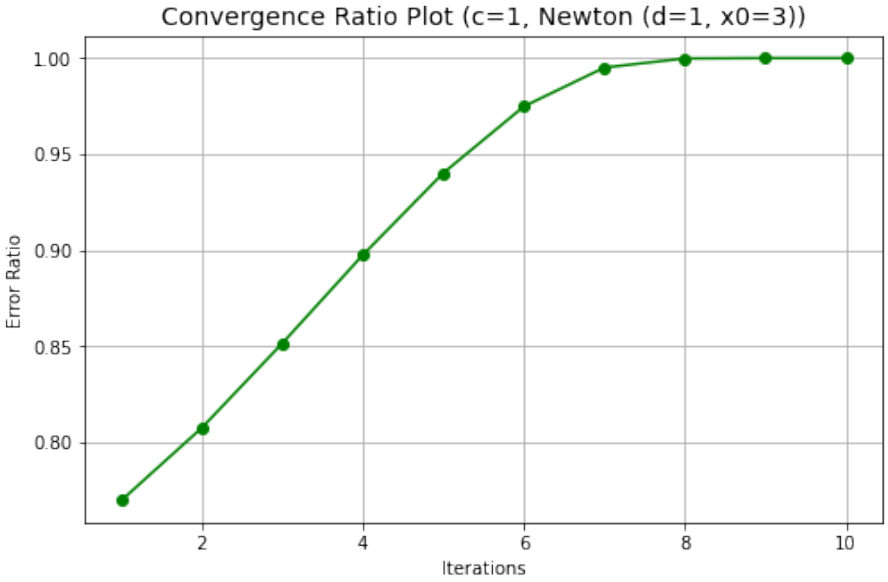


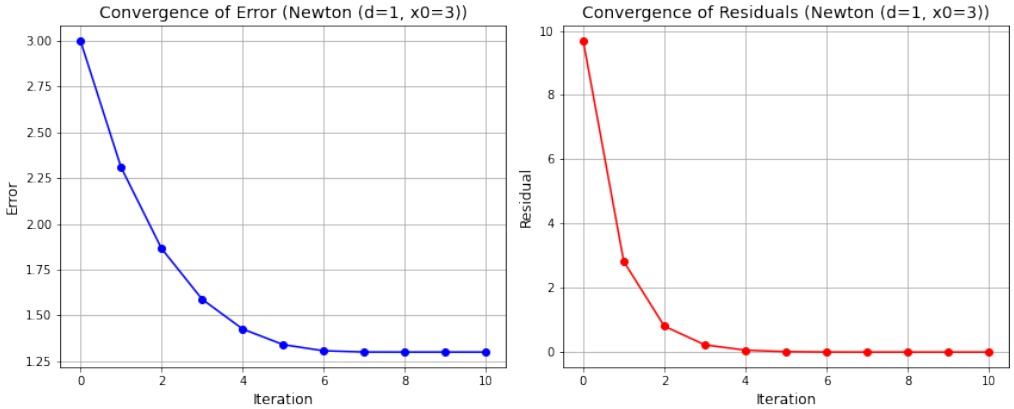




Parameter	Value
Initial Guess	3.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

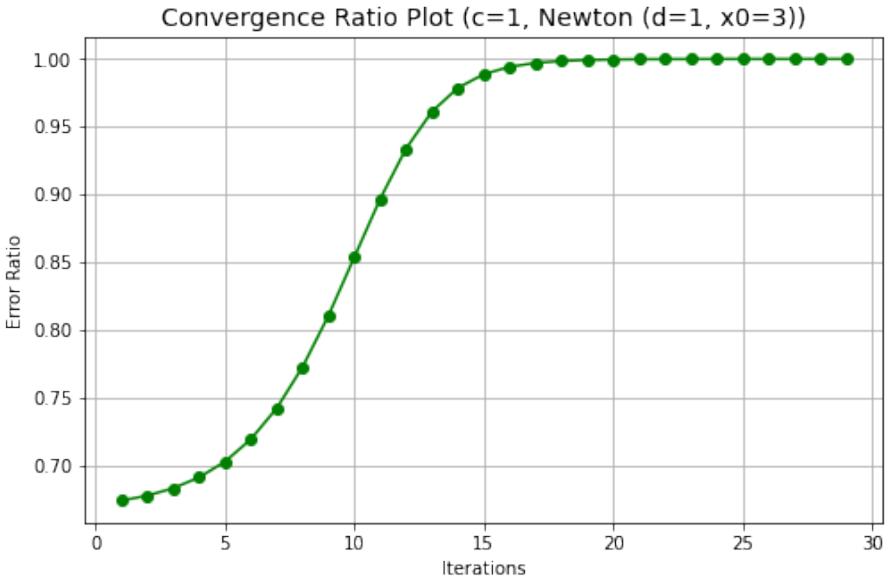
Method Used	Initial Guess	True Root	Final Error	Final Residual	Number of Iterations
Newton	3	0	1.30e+00	0.00e+00	10

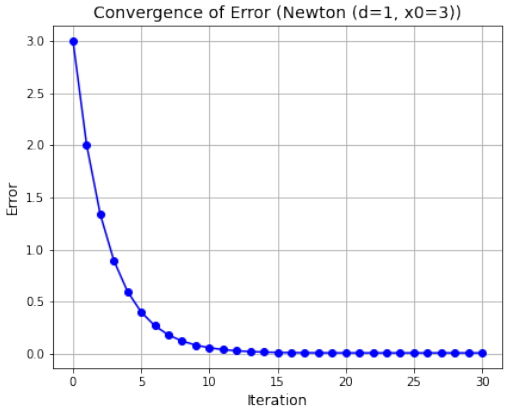


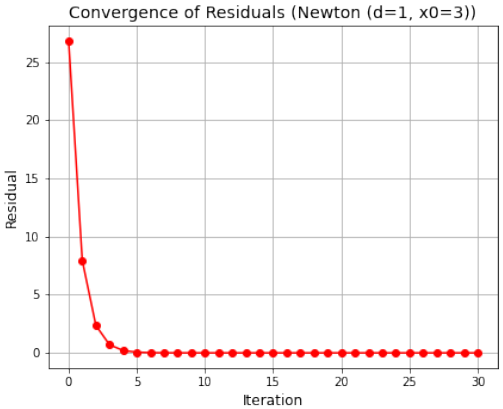


Parameter	Value
Initial Guess	3.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Final Error	Final Residual	Number of Iterations
Newton	3	0	1.30e+00	0.00e+00	10

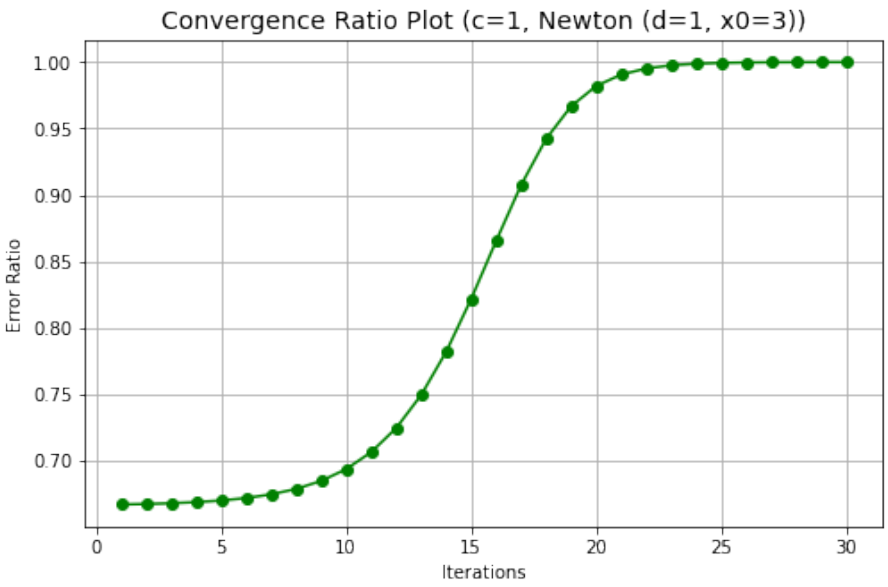




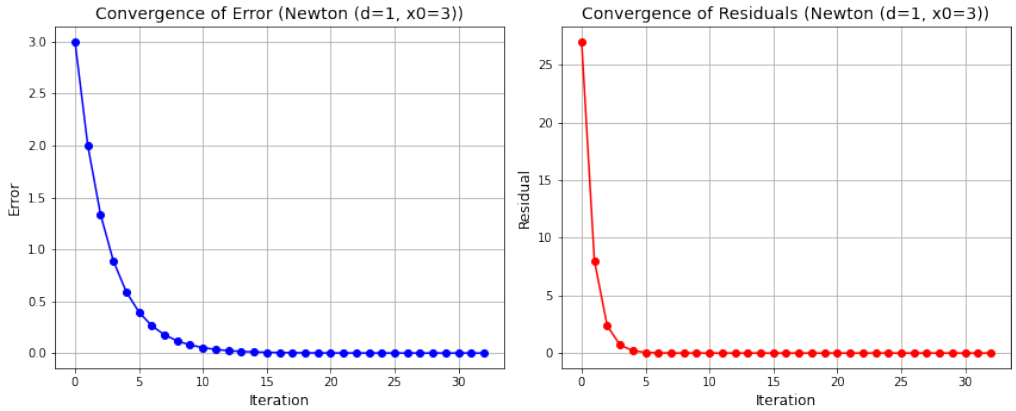


Parameter	Value
Initial Guess	3.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Final Error	Final Residual	Number of Iterations
Newton	3	0	1.00e-02	3.73e-16	30

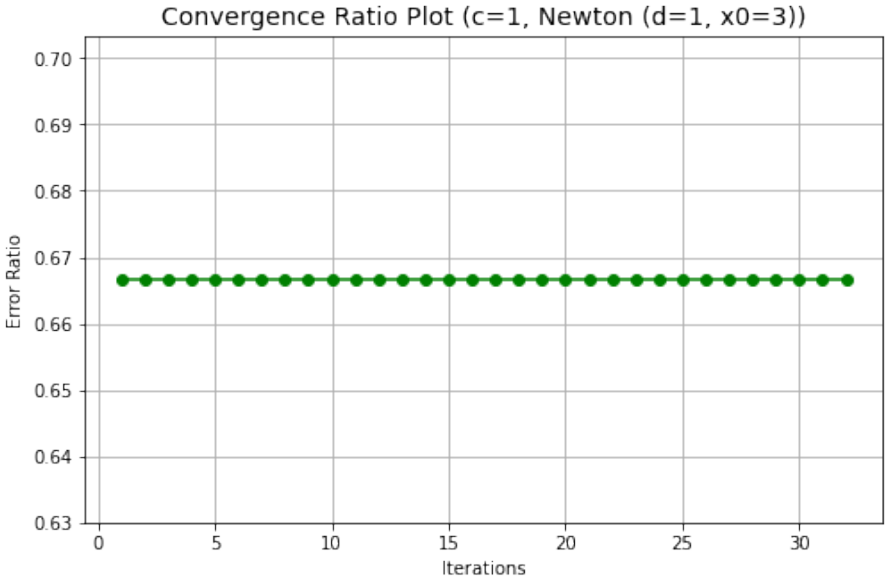


And in the limit we get:



Parameter	Value
Initial Guess	3.0
Tolerance Error	1e-08
Tolerance Residual	1e-08
Max Iterations	1000.0

Method Used	Initial Guess	True Root	Final Error	Final Residual	Number of Iterations
Newton	3	0	6.95e-06	3.36e-16	32



Conclusion: The closer we get to 0 in the limit the longer the convergence is faster. The algorithm also behaves very differently based on the position of the initial guess. In the limit, we go back to a the full loss of quadratic convergence and we get the expected worst case linear convergence for a root with multiplicity 3.

3.3 Effect on the Secant Method

How does this affect the behavior of the Secant Method?

The Secant Method behaves much more erratically given the initial point changes, but overall we observe similar convergence degradation behavior.

Conclusions

We have successfully probed an tested the various properties of different root-finding algorithms for non-linear (scalar) equations. For each method, we tested conditions and parameter settings where the algorithms behave as expected, and where there is degraded behavior in terms of convergence, speed and performance (including conditions where the algorithms fail entirely). We tested more involved conditions numerically and modeled different situations to design more comprehensive tests of theoretical hypothesis and expected capabilities of our algorithms. Each test was conducted empirically, and the results were presented in the form of metrics and graphs. Each experiment was then concluded by a explanation and a description of the results, noting peculiar or unexpected behaviors.

Appendix

We present the algorithms for each of the methods used and tested above.

Algorithm 1 Regula Falsi Method

Choose interval $[a, b]$ such that $f(a) \cdot f(b) < 0$
for $k = 0, 1, \ldots$ **do**
  $f_a \leftarrow f(a)$
  $f_b \leftarrow f(b)$
  $x_k \leftarrow \frac{a \cdot f_b - b \cdot f_a}{f_b - f_a}$
  $f_k \leftarrow f(x_k)$
  if $f_k = 0$ **then**
    Terminate: exact root found
  end if
  if $f_k \cdot f_a < 0$ **then**
    $b \leftarrow x_k$
  else
    $a \leftarrow x_k$
  end if
  if convergence criteria met **then**
    Terminate
  end if
end for

Algorithm 2 Modified Newton's Method

Choose initial guess x_0
Define multiplicity m
for $k = 0, 1, \ldots$ **do**
  $f_k \leftarrow f(x_k)$
  $f'_k \leftarrow f'(x_k)$
  if $f'_k = 0$ **then**
    Terminate: derivative is zero
  end if
  $\Delta x_k \leftarrow \frac{f_k}{m \cdot f'_k}$
  $x_{k+1} \leftarrow x_k - \Delta x_k$
  if convergence criteria met **then**
    Terminate
  end if
end for

Algorithm 3 Secant Method

Choose initial guesses x_0 and x_1
for $k = 1, 2, \ldots$ **do**
  $f_k \leftarrow f(x_k)$
  $f_{k-1} \leftarrow f(x_{k-1})$
  $\Delta x_k \leftarrow \frac{f_k(x_k - x_{k-1})}{f_k - f_{k-1}}$
  $x_{k+1} \leftarrow x_k - \Delta x_k$
  if convergence criteria met **then**
    Terminate
  end if
end for

Algorithm 4 Steffensen's Method

Choose initial guess x_0
for $k = 0, 1, \ldots$ **do**
  $f_k \leftarrow f(x_k)$
  if $f_k = 0$ **then**
    Terminate: exact root found
  end if
  $x_{temp} \leftarrow x_k + f_k$
  $g_k \leftarrow \frac{f(x_{temp}) - f_k}{f_k}$
  if $g_k = 0$ **then**
    Terminate: pathological condition
  end if
  $x_{k+1} \leftarrow x_k - \frac{f_k}{g_k}$
  if convergence criteria met **then**
    Terminate
  end if
end for