On Root Convergence Rates

By
Namir Shammas

Introduction

Numerical Analysis provides a wealth of root–seeking methods. These methods differ in basic mechanisms and strategies. While many generate a single root refinement per iteration, some of the new algorithms have adopted the approach of generating multiple root refinements per iteration. Alexander Ostrowski[1] (1893 to 1986) was a talented Russian mathematician who taught for many years at the University of Basil, Switzerland. To the best of my knowledge, he implemented the first enhancement to Newton's method that uses *two* refined guesses per iteration. Since Ostrwoski's algorithm came out, more mathematicians have adopted Ostrwoski's approach to create similar variants to Newton's method. Some of these new algorithms have more than two guess refinements per iterations.

As we pick and choose between an increasing number of available root–seeking algorithms, we pay attention to how fast they tend to converge. Since a few decades ago, mathematicians who derive new root seeking methods, also go through the elaborate derivation of equations that identify the convergence rate. The typical and general form for the convergence of an algorithm examines how fast the error (difference between the current root guess and the actual root) diminish. The general equation for convergence is:

$$e_{i+1} = K\, e_i^n \qquad (1)$$

Where the power n is the convergence rate. The value for e_i is the difference between the refined root, at iteration i, and the actual root value. The value of K is a coefficient that depends on the derivatives of the function whose root we seek and perhaps other constants and metrics. Wikipedia presents the following equation, for Newton's method, that relates consecutive error values:

$$|e_{i+1}| = \frac{|f''(\varepsilon_i)|}{2|\,f'(x_i)|} e_i^2 \qquad (1b)$$

Where ε_i is a value located between x_i and the actual root r. Equation 1b gives you an example what coefficient K in equation 1 looks like for Newton's method. The

same equation also suggest that K is a function of the current refined root value and contributes, along with e_i^2 to the value of the new error. The term e_i^2 suggest that Newton's method has a quadratic convergence rate.

A Change in Perspective

I must admit that my prior view of the convergence rate as shown in equation 1 has been very naïve. I had assumed that after a few initial iterations, the errors in the refined root values adhere closely to the convergence rate order. This adherence represents a steady state that the iteration process reaches quickly. This steady state can happen if the values of the first and second derivatives don't change by much WHEN the initial guess for the root is very close to the root. Choose an initial guess for the root that is far from the root, and all bets are off! Recently, I developed the new Ostrowski–Halley algorithm using Ostrowski's approach to enhance Halley's method. Several colleagues, on the hpmuseum.com web site, asked me what the convergence rate was for my new method. I did not know, since I did not follow the typical elaborate mathematical derivation for the convergence rate. Instead, I compared the number of iterations and function calls of my new algorithm with those of Newton, Halley, and Ostrowski. The results for a dozen test functions showed that the new algorithm did better than the other three with most of the test functions. Since the results of my new algorithm were reasonably (but not spectacularly) better than the results from Halley, I assumed that my new algorithm has a cubic convergence rate, just like Halley's method. I had also assumed, until recently, that Ostrowski's method also had a cubic convergence rate. Several colleagues pointed out that Ostrowski's method had a fourth order convergence rate.

The query from my Internet colleagues drove me to dive deeper into studying in more details the progress in the calculated errors associated with the root refinements. I decided that the best approach is in using functions where I know the exact roots values, or can easily calculate it. This way, I can accurately calculate the error for each iteration. The purpose of this study is:

- Determine the *effective* convergence rate for an *entire* root–seeking set of calculations.
- Determine the range of iterations when the effective convergence rate matches or comes close to the theoretical convergence rate.
- Learn to empirically determine or verify the effective and theoretical convergence rates for root–seeking algorithms.

Background Theory

A colleague recently sent me a link to an article written by Thukral[2]. The author mentions a method to estimate the convergence rate using:

$$n = \ln|\delta_i / \delta_{i-1}| / \ln|\delta_{i-1} / \delta_{i-2}| \quad (2)$$

Where

$$\delta_i = f(x_i) / f'(x_i) \quad (3)$$

Based on Newton's method (i.e., $x_{i+1} = x_i - f(x_i) / f'(x_i)$), we can rewrite equation 3 as,

$$\delta_i = x_i - x_{i+1} \quad (3b)$$

Equation 3b makes δ_i depend on two consecutive refinements of the roots. I suggest the following modification that reduces the reliance on refined root values:

$$\delta_i = x_i - r = e_i \quad (4)$$

Where r is the exact root OR the *highly refined* root value that meets strict tolerance values. Equation 4 calculates the actual error of the refined root values, when the exact root is known. Thus, equation 2 can be written as:

$$n = \max(\ln|e_i / e_{i-1}| / \ln|e_{i-1} / e_{i-2}|) \text{ for } i=1, 2, 3, \ldots \quad (5)$$

I am using the function max to point out that n is the maximum value of the logarithmic ratios calculated for the various iterations.

Looking back at equation 2 and 3, if we assume that the slope of consecutive root refinements does not change much in three iterations, we can take the derivatives out of the equation and write another equation to estimate n:

$$n = \max(\ln|f_i / f_{i-1}| / \ln|f_{i-1} / f_{i-2}|) \text{ for } i=1, 2, 3, \ldots \quad (6)$$

Like with equation 5, I am using the function max to point out that n is the maximum value of the logarithmic ratios calculated for the various iterations.

In the case of using a whole array of error values, we can calculate the slope, using linear regression, of the linearized form of equation 1:

$$\ln(e_{i+1}) = \ln(K) + + n \ln(e_i) \quad (7)$$

The slope in equation 7 is n, the effective convergence rate. The intercept is ln(K). Armed with equations 5, 6, and 7, we can study the convergence rate of a few algorithms. Equation 7 gives us an overall convergence rate for the iterations set. Equations 5 and 6 gives the maximum value of the individual estimates for the convergence rate. Expect the estimated convergence rate values to be closer to the theoretical value in the final stages of the iteration process.

I will also study how equations 5, 6, and 7, work with the selected algorithms using the following schemes:

1. Use finite difference approximations to estimate the function's derivatives as needed by each algorithm. This approach is applied with problems whose roots are known or can be easily calculated.
2. Accurately calculate the function derivatives, as needed. This approach is applied with problems whose roots are known or can be easily calculated.
3. Use finite difference approximations to estimate the function's derivatives as needed by each algorithm. This approach is applied with problems whose roots are NOT accurately known. This approach calculates the refined root that meets the established tolerance value and then back calculates the errors between the previous root refinements and the last one. I will apply this approach once, with Newton's method, to give you an idea of the effect of using the last refined root as an estimate for the real root.

Schemes 1 and 3 examine the effect of approximating the derivatives and compare the results with scheme 2 where the derivatives are accurately calculated. In addition, schemes 1 and 2 compare using test functions with exact known roots, compared with scheme 3 where the exact roots are not known. Both sets of comparisons examine the effect of using approximations versus working with the exact values.

About the Calculations

I used the following equations to estimate the convergence rate for various algorithms:

1. f(x) = exp(x–3) + 3*(x–3)^2 – 1, which has a root at x=3.
2. f(x) = (x–3) * (x–20) * (x–50), which has roots at x=3, 20, and 50.
3. f(x) = x^3 + ln(x+1), which has a root at x = 0.
4. f(x) = exp(x) –100, which has a root at x=ln(100).

Figures 1 to 3 show the plots for the first three functions.

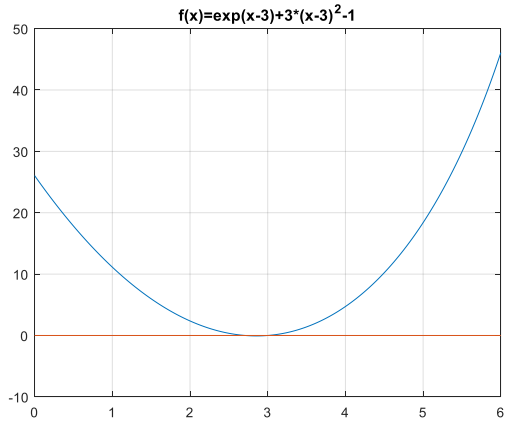


Figure 1. The function f(x) = exp(x–3)+3(x–3)^2–1.*

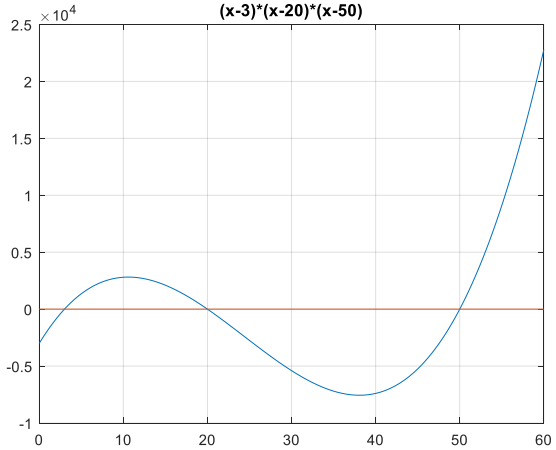


Figure 2. The function f(x) = (x–3)(x–20)*(x–50).*

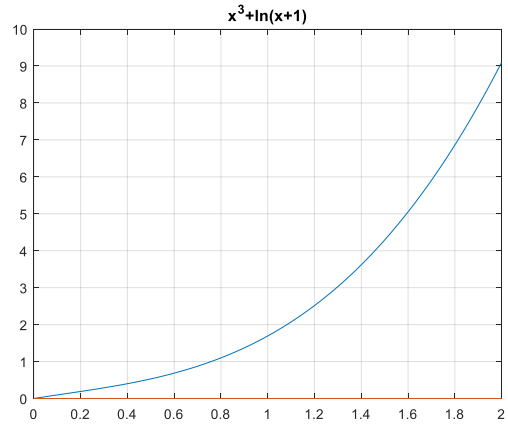


Figure 3. The function f(x) = x^3+ln(x+1).

I chose the last function since it is a smooth function, has no minima or maxima, has simple first and second derivatives (both equal to exp(x)), and has a root that can easily and accurately calculated. I did not feel that the study needed dozens of test functions, since I was out to test that the theoretical convergence rate does not hold for a significant portion of the iterations. A few examples that make the point are adequate.

I carried out the calculations for the roots of different algorithms using a tolerance value of 10^{-9}. I used Excel workbooks and coded the programs in VBA. The spreadsheets allow me to enter the initial guess, the actual root, the tolerance value, and mathematical expressions for f(x) and its derivatives (in the cases where I need to accurately calculate the derivatives). These expressions permit me to work with different functions without having to hard code then in the VBA source code. I applied linear regression to the array of ln(error) values. In several cases, I had to exclude the last value where successive errors are (almost) equal (and where the best refined root value occurs) to avoid computational error. As for using equations 5 and 6, the values generated by these equations were, for the leading iterations, generally close to 1. The maximum logarithmic ratios generated by the equations usually occurred near the end of the iterations. These are the values that theory predicts, but is very disappointing to see that the theoretical convergence rate occurs as a *blip* in the entire process! I was very surprised to see a drop in the convergence rate at the last one or two iterations!

The convergence rate values for the results shown in all of the next tables are all rounded to two decimal places.

Newton's Method

Table 1 shows the results for Newton's algorithm that approximates the derivatives. Newton's method has a quadratic convergence rate. Table 1 shows that the overall convergence rate by applying linear regression to equation 7 yields a linear order. The last two columns of the table shows the results of using equation 5 and 6. The values do reach second order convergence, matching the theoretical convergence rate. There is one value that hints at cubic convergence rate, which I consider as an outlier. Applying different initial guess to different functions does affect the estimated convergence rates.

f(x)	Root	Initial Guess	Slope of Logarithmic Error Ratios (eqn. 7)	Max ln(err) Ratios (eqn. 5)	Max ln(f) Ratios (eqn. 6)
exp(x–3) + 3*(x–3)^2 – 1	3	10	1.22	1.39	1.18
exp(x–3) + 3*(x–3)^2 – 1	3	7	1.12	1.33	1.16
(x–3) * (x–20) * (x–50)	3	–100	1.30	1.63	1.77
(x–3) * (x–20) * (x–50)	50	100	1.13	1.64	1.50
x^3 + ln(1+x)	0	5	1.31	2.36	2.02
x^3 + ln(1+x)	0	10	1.29	2.94	2.22
exp(x) – 100	ln(100)	20	1.17	1.74	1.52
exp(x) – 100	ln(100)	50	1.18	1.68	1.55

Table 1. Results for Newton's algorithm that approximates the derivatives.

Table 2 shows the results for Newton's algorithm that calculates exact derivatives. The slope of equation 7 is closer to 2 in table 2, than in table 1. The estimates for the convergence rate using equations 5 and 6 are right on target. Personally, I was very surprised at the general degradation of the convergence rate when using approximations to the first derivative with Newton's method. I have always considered that approach as practical and was unaware of the potential loss of convergence rate!

f(x)	Root	Initial Guess	Slope of Logarithmic Error Ratios (eqn. 7)	Max ln(err) Ratios (eqn. 5)	Max ln(f) Ratios (eqn. 6)
exp(x–3) + 3*(x–3)^2 – 1	3	10	1.34	2.00	2.00
exp(x–3) + 3*(x–3)^2 – 1	3	7	1.63	2.00	1.99
(x–3) * (x–20) * (x–50)	3	–100	1.47	2.00	2.00
(x–3) * (x–20) * (x–50)	50	100	1.30	2.00	2.00
x^3 + ln(1+x)	0	5	1.39	2.00	2.00

f(x)	Root	Initial Guess	Slope of Logarithmic Error Ratios (eqn. 7)	Max ln(err) Ratios (eqn. 5)	Max ln(f) Ratios (eqn. 6)
x^3 + ln(1+x)	0	10	1.75	1.99	1.99
exp(x) – 100	ln(100)	20	1.15	2.00	1.99
exp(x) – 100	ln(100)	50	1.23	1.99	1.99

Table 2. Results for Newton's algorithm that calculates exact derivatives.

Table 3 shows the results for Newton's algorithm that calculates roots for a different set of functions that have no exact values. The functions are:

1. f(x) = exp(x)–3*x^2
2. f(x) = (x–1/0.3)*(x–1/0.0511)*(x–1/0.02345)
3. f(x) = x^3+ln(1/0.9+x)
4. f(x) = exp(x)–1/0.00111

The above functions are similar to the set of functions with exact roots. The similarity is intentional in order to enable better comparison with the other results.

The values in the table are similar to those of table 1 for the result of equations 5 and 7. In the case of equation 6, the results hint at a (rounded) convergence rate are mostly close to 1.5. There are values that hint at orders 2 to 5 (if we round up).. These results lead me to conclude that attempting to calculate the convergence rate using functions whose roots are not know exactly, can lead to unreliable results!

f(x)	Root	Initial Guess	Slope of Logarithmic Error Ratios (eqn. 7)	Max ln(err) Ratios (eqn. 5)	Max ln(f) Ratios (eqn. 6)
exp(x)–3*x^2	3.7330	10	1.08	1.62	1.44
exp(x)–3*x^2	3.7330	7	1.14	1.55	1.40
(x–1/0.3)*(x–1/0.0511)*(x–1/0.02345)	3.3333	–100	1.30	2.25	2.21
(x–1/0.3)*(x–1/0.0511)*(x–1/0.02345)	42.6439	100	1.13	1.60	1.48
x^3+ln(1/0.9+x)	–0.10978	5	1.23	2.73	1.82

f(x)	Root	Initial Guess	Slope of Logarithmic Error Ratios (eqn. 7)	Max ln(err) Ratios (eqn. 5)	Max ln(f) Ratios (eqn. 6)
x^3+ln(1/0.9+x)	–0.10978	10	1.25	4.53	3.36
exp(x)–1/0.00111	6.80339	20	1.18	1.70	1.47
exp(x)–1/0.00111	6.80339	50	1.20	1.70	1.46

Table 3. Results for Newton's algorithm that calculates roots that have no exact values.

Halley's Method

Table 4 shows the results for Halley's algorithm that approximates the derivatives. This algorithm has a cubic convergence rate. The values for the slope of equation 7 are in the range of linear and quadratic. The results of equations 5 and 6 do favor a cubic convergence rate.

f(x)	Root	Initial Guess	Slope of Logarithmic Error Ratios (eqn. 7)	Max ln(err) Ratios (eqn. 5)	Max ln(f) Ratios (eqn. 6)
exp(x–3) + 3*(x–3)^2 – 1	3	10	1.8	2.53	2.26
exp(x–3) + 3*(x–3)^2 – 1	3	7	1.31	2.33	2.20
(x–3) * (x–20) * (x–50)	3	–100	1.69	2.86	2.73
(x–3) * (x–20) * (x–50)	50	100	1.23	2.41	1.90
x^3 + ln(1+x)	0	5	1.25	3.18	3.26
x^3 + ln(1+x)	0	10	1.32	3.18	3.27
exp(x) – 100	ln(100)	20	1.37	2.81	1.98
exp(x) – 100	ln(100)	50	1.98	2.83	2.08

Table 4. Results for Halley's algorithm that approximates the derivatives.

Table 5 shows the results for Halley's algorithm that calculates exact derivatives. As with table 2, equations 5 and 6 give results that are indicative of a cubic convergence rate. In the case of the slope of equation 7, only one value goes

slightly above 2.5. The result of the column shows overall convergence rates that are linear or quadratic.

f(x)	Root	Initial Guess	Slope of Logarithmic Error Ratios (eqn. 7)	Max ln(err) Ratios (eqn. 5)	Max ln(f) Ratios (eqn. 6)
exp(x–3) + 3*(x–3)^2 – 1	3	10	2.53	2.93	2.36
exp(x–3) + 3*(x–3)^2 – 1	3	7	2.36	2.82	2.66
(x–3) * (x–20) * (x–50)	3	–100	2.09	2.90	2.76
(x–3) * (x–20) * (x–50)	50	100	0.94	2.95	2.88
x^3 + ln(1+x)	0	5	1.18	3.10	3.17
x^3 + ln(1+x)	0	10	1.20	2.99	2.98
exp(x) – 100	ln(100)	20	1.53	3.00	2.93
exp(x) – 100	ln(100)	50	1.54	3.00	2.93

Table 5. Results for Halley's algorithm that calculates exact derivatives.

Potra–Ptak Method

I randomly picked this additional method since it appears in the article by Thukral[2]. The algorithm is said to have a cubic convergence rate and is a variant of Ostrowski's method. This algorithm uses the following equations to refine the guesses for the root:

$$y_n = x_n - \frac{f(x_n)}{f'(x_n)} \tag{8}$$

$$x_{n+1} = x_n - \frac{(f(x_n)+f(y_n))}{f'(x_n)} \tag{9}$$

Table 6 shows the results for the Potra–Ptak algorithm that calculates exact derivatives. Most of the results of equations 5 and 6 do reflect a cubic convergence rate. A few outliers suggest higher convergence rates. We can safely ignore these values. The slope of equation 7 varies between linear and quadratic overall convergence rate.

f(x)	Root	Initial Guess	Slope of Logarithmic Error Ratios (eqn. 7)	Max ln(err) Ratios (eqn. 5)	Max ln(f) Ratios (eqn. 6)
exp(x–3) + 3*(x–3)^2 – 1	3	10	1.97	2.84	2.74
exp(x–3) + 3*(x–3)^2 – 1	3	7	2.22	2.79	2.66
(x–3) * (x–20) * (x–50)	3	–100	1.77	2.94	2.89
(x–3) * (x–20) * (x–50)	50	100	0.98	2.94	2.96
x^3 + ln(1+x)	0	5	1.375	4.75	4.69
x^3 + ln(1+x)	0	10	1.42	6.29	4.76
exp(x) – 100	ln(100)	20	1.59	2.98	2.96
exp(x) – 100	ln(100)	50	1.41	2.99	2.97

Table 6. Results for the Potra–Ptak algorithm that calculates exact derivatives.

Ostrowski's Method

Table 7 shows the results for Ostrowski's algorithm that approximates the derivatives. The results of equation 7 shows that the overall convergence rate varies between an order of 1 and 4/3. The results from using equations 5 and 6 reflect convergence rates that are in the 3 to 4 range. Thus the results match the fourth order convergence rates that was reported to me by a colleague.

f(x)	Root	Initial Guess	Slope of Logarithmic Error Ratios (eqn. 7)	Max ln(err) Ratios (eqn. 5)	Max ln(f) Ratios (eqn. 6)
exp(x–3) + 3*(x–3)^2 – 1	3	10	1.07	2.69	1.00
exp(x–3) + 3*(x–3)^2 – 1	3	7	1.04	2.55	1.73
(x–3) * (x–20) * (x–50)	3	–100	1.30	2.93	2.57
(x–3) * (x–20) * (x–50)	50	100	1.11	1.90	1.67
x^3 + ln(1+x)	0	5	1.37	3.74	3.53
x^3 + ln(1+x)	0	10	1.38	3.49	3.46

f(x)	Root	Initial Guess	Slope of Logarithmic Error Ratios (eqn. 7)	Max ln(err) Ratios (eqn. 5)	Max ln(f) Ratios (eqn. 6)
exp(x) – 100	ln(100)	20	1.31	2.72	1.75
exp(x) – 100	ln(100)	50	1.31	1.95	1.59

Table 7. Results for Ostrowski's algorithm that approximates the derivatives.

Table 8 shows the results for Ostrowski's algorithm that uses exact derivatives. Keep in mind that the core Ostrowski algorithm estimates derivatives that used the intermediate root refinement. Nevertheless, the values of the slope in equation 7 are both higher (near 2) and lower (near 0.9) compared to similar results in table 7. The results of equations 5 and 6 support a fourth order convergence rate for the Ostrowski method.

f(x)	Root	Initial Guess	Slope of Logarithmic Error Ratios	Max ln(err) Ratios	Max ln(f) Ratios
exp(x–3) + 3*(x–3)^2 – 1	3	10	2.48	3.27	2.76
exp(x–3) + 3*(x–3)^2 – 1	3	7	2.11	3.36	2.89
(x–3) * (x–20) * (x–50)	3	–100	2.13	3.59	3.17
(x–3) * (x–20) * (x–50)	50	100	0.83	3.57	3.10
x^3 + ln(1+x)	0	5	0.94	3.62	3.39
x^3 + ln(1+x)	0	10	1.01	3.93	3.88
exp(x) – 100	ln(100)	20	1.12	3.85	3.16
exp(x) – 100	ln(100)	50	1.34	3.97	3.79

Table 8. Results for Ostrowski's algorithm that uses exact derivatives.

Ostrowksi–Halley Method

Table 9 shows the results for my new Ostrowski–Halley algorithm. When I was asked about the convergence rate I gave a guess of 3. The table shows that the results from using equations 5 and 6 hint at orders of 4 or 5. I will go with 5, since it was also a colleague's estimated value. The results of the slopes using equation 7 shows mostly linear rates with one value above cubic convergence rate. I think the verdict for the convergence rate for this new algorithm is an order of 5.

f(x)	Root	Initial Guess	Slope of Logarithmic Error Ratios (eqn. 7)	Max ln(err) Ratios (eqn. 5)	Max ln(f) Ratios (eqn. 6)
exp(x–3) + 3*(x–3)^2 – 1	3	10	1.29	3.52	3.12
exp(x–3) + 3*(x–3)^2 – 1	3	7	1.32	3.52	2.63
(x–3) * (x–20) * (x–50)	3	–100	3.39	4.28	4.08
(x–3) * (x–20) * (x–50)	50	100	1.21	3.87	3.01
x^3 + ln(1+x)	0	5	1.36	4.85	4.98
x^3 + ln(1+x)	0	10	1.31	4.86	4.98
exp(x) – 100	ln(100)	20	1.73	2.26	1.21
exp(x) – 100	ln(100)	50	1.40	4.00	3.39

Table 9. Results for Ostrowski–Halley algorithm that approximates the derivatives.

Table 10 shows the results for my new Ostrowski–Halley algorithm. The table shows that the results from using equations 5 and 6 hint at orders of 4 or 5. The latter value is also a colleague's estimated value. The results of the slopes using equation 7 shows mostly quadratic and cubic convergence rates. Again, the verdict for the convergence rate for this new algorithm is an order of 5.

f(x)	Root	Initial Guess	Slope of Logarithmic Error Ratios (eqn. 7)	Max ln(err) Ratios (eqn. 5)	Max ln(f) Ratios (eqn. 6)
exp(x–3) + 3*(x–3)^2 – 1	3	10	1.36	3.74	3.53
exp(x–3) + 3*(x–3)^2 – 1	3	7	0.93	3.65	3.49
(x–3) * (x–20) * (x–50)	3	–100	3.43	4.28	4.10
(x–3) * (x–20) * (x–50)	50	100	3.32	3.53	2.78
x^3 + ln(1+x)	0	5	2.21	4.73	4.83
x^3 + ln(1+x)	0	10	2.24	4.72	4.82
exp(x) – 100	ln(100)	20	1.94	2.73	1.46

exp(x) – 100	ln(100)	50	2.66	4.00	3.56

Table 10. Results for Ostrowski–Halley algorithm that calculates the derivatives.

Conclusion

The advantage in determining convergence rate of a root–seeking method is that we have the luxury of choosing the test functions. The use of equations 5 and 6 to estimate convergence rates proved to be practical. Equation 5 is more robust than equation 6 since it simply uses the differences between the current root refinement and the exact root. Equation 6, while still useful, relies on the assumption that the function derivative does not change much in value. You can regard equation 6 as a way to doublecheck on the results of equation 5. These equations work better with root–seeking implementations that follow these rules:

1. Calculate the roots of functions with exact roots.
2. Calculate the exact derivatives required in the algorithm used.

I did notice that function f(x) = x^3 + ln(1+x) tends to give high estimates for the convergence rate. These values sometimes exceed the theoretical convergence rate. When I tried in investigate if a variant of that function, say, f(x)=(x–3)^3+ln(x–2), did better, I got convergence rates very close to 1 using equations 5 and 6, and about 0.92 using equation 7. Thus, this variant function went to the other extreme! Therefore, I would say the jury is not out yet on function f(x) = x^3 + ln(1+x).

Using equation 7 shows the actual and effective convergence rate for the iterations, given a function f(x) and the parameters associated with the solution. As such, equation 7 gives you the big picture. This big picture depends on, among many factors, the initial root guess. You can repeat the calculations with initial root guesses that are closer to the root and reexamine the effective convergence rate.

The theoretical convergence rates supplied for various algorithms are valid for a few iterations as the refined roots approach the actual root. This limitation is very disappointing and downplays the importance of the theoretical convergence rates. If the initial guesses for the root are far from the root, the effective and overall convergence rate is generally of order one, and at the very best, one order lower than the theoretical convergence rate. It seems that the value of coefficient K, in equation 1, plays a more relevant role in calculating the reduction in the errors associated with the refined root values. In solving for the root of a nonlinear function, the effective convergence rate is influenced by the following factors:

- The value of the initial guess for the root.
- The actual number of roots and their values.
- The values of the function and its derivatives (i.e. the function's curvature) at various iterations.
- The changes and variations in the function slope and curvature around the targeted root and possibly other neighboring roots.

The above factors make it difficult to create simplified rules of thumb for estimating or verifying the convergence rate of a function.

This study also *attempts* to answers the question regarding the convergence rate for my new algorithm. It has a fifth order if you include the results of f(x) = x^3 + ln(1+x). Otherwise, the convergence rate is fourth order.

I have dealt with root–seeking algorithms for over four decades. This study that I conducted is a real eye–opener for me. The dynamics of convergence and its rate exhibit enough variations to prevent using the theoretical convergence rates as a robust predictor of the lifetime of the iterations that lead to a root. I consider the overall convergence rate as far more relevant and meaningful that the theoretical convergence rates that play a very limited role in the lifetime of the entire iterative process. I have always followed my gut feeling in working case by case and observing the number of iterations and function call as a good indication of the performance of an algorithm. Now I have new tools to empirically determine the effective and theoretical convergence rate of root–seeking algorithms. In the future, I plan revise the selected functions in favor of a collection of more *well behaved* functions. These functions should closely agree with the theoretical convergence rates for as many and diverse root-seeking algorithms as possible.

Using the Excel Files

The RootCR.ZIP file which contains this document also includes a number of macro–enabled Excel files. In each file, you find the VBA programs in the *ThisWorksheet* macro *project* located in the *Project Explorer* pane of the VBE (Visual Basic Editor) window, as shown in Figure 4. The Project Explorer has several spreadsheet-associated modules in addition to a general *ThisWorksheet* macro project. This macro project contains subroutines that work with the currently selected worksheet, and therefore can work with any worksheet. The VBE editor shows other projects associated with existing worksheets. I have kept these projects empty of macros. Placing subroutines in these macro projects will

allow you to run the code that interacts only with its associated worksheet. The Excel files, I am providing, have all the details which calculate the convergence rates for different algorithms and different ways to calculate the derivatives.

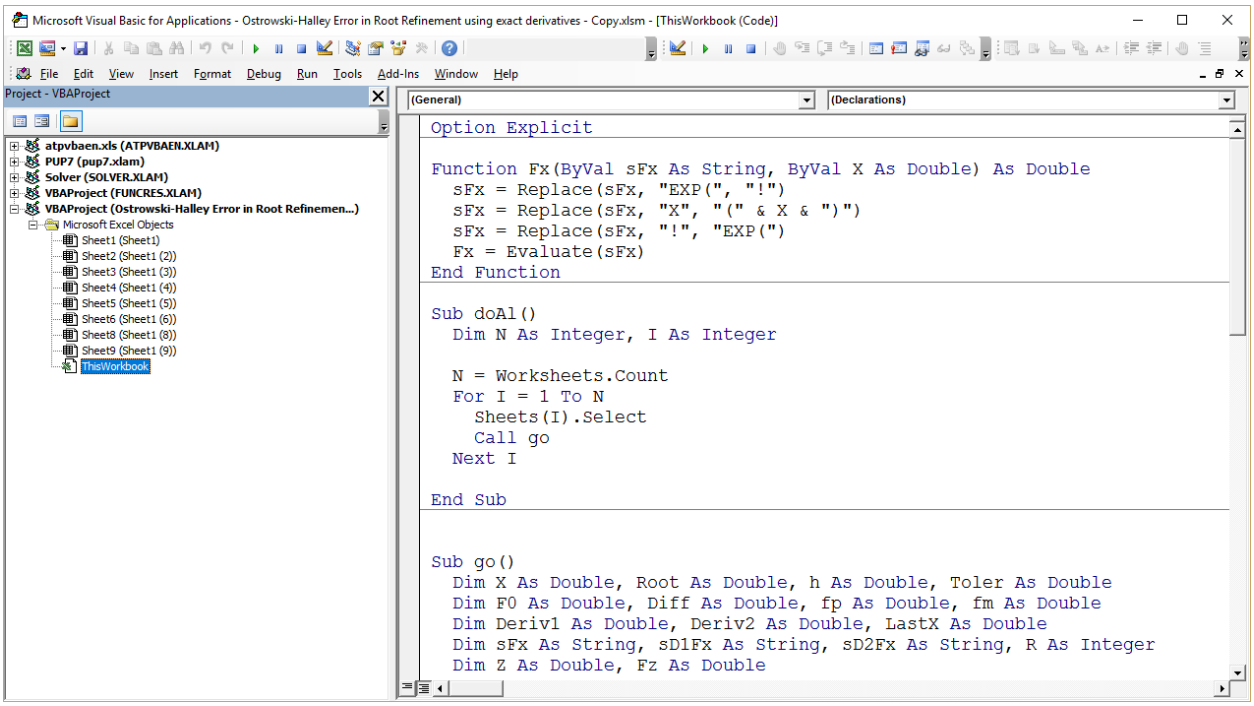
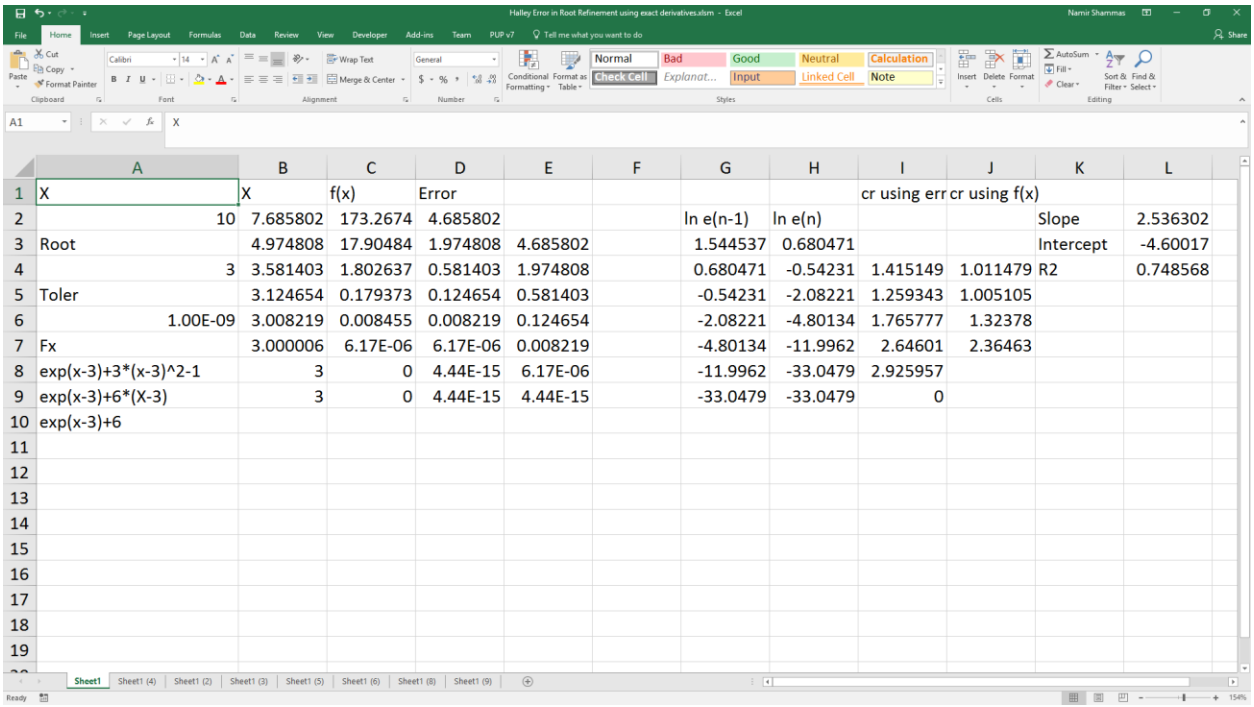


Figure 4. The VBE window showing the project explorer in the left window and the VBA code for ThisWorksheet macro project.

Figure 5 shows a typical worksheet used in the calculations. All of the spreadsheets use the following cells for input:

1. Cell A2 contains the initial guess for the root.
2. Cell A4 contains the exact root. Examples for this input are **3**, **–100**, and **=LN(100)**. The latter example shows how to calculate an exact root.
3. Cell A6 contains the tolerance for the root.
4. Cell A8 contains the expression for the function f(x). An example of input is the text **exp(x)–3*x^2**. The expression is case insensitive.
5. Cells A9 contains the expression for the first derivative of f(x), when needed. An example of input is the text **exp(x)–6*x**. The expression is case insensitive.
6. Cells A10 contains the expression for the second derivative of f(x), when needed. An example of input is the text **exp(x)–6**. The expression is case insensitive.



	A	B	C	D	E	F	G	H	I	J	K	L
1	X	X	f(x)	Error					cr using err	cr using f(x)		
2	10	7.685802	173.2674	4.685802			ln e(n-1)	ln e(n)			Slope	2.536302
3	Root	4.974808	17.90484	1.974808	4.685802		1.544537	0.680471			Intercept	-4.60017
4	3	3.581403	1.802637	0.581403	1.974808		0.680471	-0.54231	1.415149	1.011479	R2	0.748568
5	Toler	3.124654	0.179373	0.124654	0.581403		-0.54231	-2.08221	1.259343	1.005105		
6	1.00E-09	3.008219	0.008455	0.008219	0.124654		-2.08221	-4.80134	1.765777	1.32378		
7	Fx	3.000006	6.17E-06	6.17E-06	0.008219		-4.80134	-11.9962	2.64601	2.36463		
8	exp(x-3)+3*(x-3)^2-1	3	0	4.44E-15	6.17E-06		-11.9962	-33.0479	2.925957			
9	exp(x-3)+6*(X-3)	3	0	4.44E-15	4.44E-15		-33.0479	-33.0479	0			
10	exp(x-3)+6											

Figure 5. A typical worksheet used in the calculations.

To perform the calculations on the currently selected worksheet you need to first open the VBE window editor (by pressing the Alt-F11 keys) and then execute the macro SUB go() located in the *ThisWorksheet* macro project. To perform the calculations for all the worksheets, in one swoop, you need to execute macro SUB doAll() in the *ThisWorksheet* macro project.

If you want to test other algorithms and/or use your own test functions, follow these general steps:

1. Make a copy of one the Excel files that I provide you, and give it an appropriate new name.
2. Invoke the VBE editor (by pressing the Alt-F11 keys) and locate the VBA code in the *ThisWorksheet* macro project.
3. To test different test functions, you can edit the values in cells A2, A4, A6, A8, and possibly A9 and A10. Depending on the number of test functions you plan to use, you can either delete worksheets (if you plan to use less than eight test functions) or easily add new ones by making copies of existing worksheets.
4. To test different root–seeking algorithms, edit the code in the subroutine go(). You may need to insert ON ERROR GOTO error handlers to allow both macros SUB go() and SUB doAll() to operate smoothly.

5. Once you are done, use the macro SUB go() to recalculate the root for the currently selected worksheet. Alternatively, you can execute the macro SUB doAll() to recalculate the roots in all of your worksheets. To run either macro, place the mouse cursor anywhere inside the code of the targeted macro and then click the run button (looks like a left-arrow pointing play button) located at the top icon bar. Figure 4 actually shows two run buttons in the top icon bar. For further assistance with running Excel VBA code, consult the Internet or any Excel VBA programming book.

Reference

1. N. Shammas, **Ostrowski's Method for Finding Roots**, HP Solve #28. You can find the pdf file by Internet search.
2. R. Thukral, **New Modification of Newton Method with Third–Order Convergence for Solving Nonlinear Equations of Type f(0)=0**, American Journal of Computational and Applied Mathematics, p–ISSN: 2165–8935, e–ISSN: 2165–8943, 2016; 6(1): 14–18.
3. A. M. Ostrowski, **Solution of Equations and Systems of Equations**, Academic Press; second edition (1966).
4. Newton's Method, Wikipedia at https://en.wikipedia.org/wiki/Newton%27s_method.
5. Halley's Method, Wikipedia at https://en.wikipedia.org/wiki/Halley%27s_method.

Document History

Date	*Version*	*Comments*
1/30/2017	1.0	Initial release.

