The Super & Hyper Secant Root-Seeking Algorithm

By

Namir Shammas

Introduction

Ostrowski was a Russian mathematician who taught for many years at the University of Basil, Switzerland. He proposed an enhancement to Newton's root seeking algorithm. Ostrowski suggested a new twist such that each iteration offers two refinements for the root—one of them being intermediate. The Ostrowski algorithm matches Halley's root-seeking algorithm in its third order rate of convergence. Recently, the Ostrowski algorithm inspired many mathematicians to device root-seeking algorithms with two or more refinements to the root per iteration.

I recently applied Ostrowski's approach to Halley's method and was able to produce a good working algorithm. I decided to further experiment with applying Ostrowski's *basic approach* to the secant method, which itself is a rough and good approximation of Newton's method. I was able to generate two algorithms—the Super Secant and the Hyper Secant methods.

Legacy Root-Seeking Algorithms

In this section, we briefly discuss the root-seeking methods of Newton, Halley, and Ostrowski. If you are already familiar with these algorithms you can skip to the section that describes the new algorithms.

The Newton Method

One of the most popular root-seeking algorithms is the Newton method (also called the Newton-Raphson method). While Isaac Newton had little to do with the algorithm in its current form, it was Thomas Simpson (better known for Simpson's rule for numerical integration) who gave it its name and homage to Sir Isaac Newton.

The equation for Newton's method that refined a guess for the root is:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \qquad (1)$$

Equation 1 requires evaluating the function f(x) and it's derivative f'(x) which can be approximated using the forward difference approximation:

$$f'(x) \simeq (f(x+h) - f(x))/h \quad (2)$$

Where h = 0.02(1 + |x|). Newton's method usually converges at a second order rate.

The Halley Method

Halley devised a method for calculating roots that has a third order convergence rate. The root-refining equation for this algorithm is:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}\left[1 - \frac{f(x_i)}{f'(x_i)}\frac{f''(x_i)}{2f'(x_i)}\right]^{-1} \quad (3)$$

Or,

$$x_{i+1} = x_i - \frac{2f(x_i)f'(x_i)}{2[f'(x_i)]^2 - f(x_i)f''(x_i)} \quad (3b)$$

The first and second derivatives are calculated using the following central difference approximations:

$$f'(x) \simeq (f(x+h) - f(x-h))/2h \quad (4)$$

$$f''(x) \simeq (f(x+h) - 2f(x) + f(x-h))/h^2 \quad (5)$$

The Ostrowski Method

While relatively newer than the previous algorithms, I am including the Ostrowski method in this section, since it is a few decades old. The Ostrowski method generates two refinements for the root in each iteration, the first refinement is an intermediate one. The method uses the following two equations:

$$y_i = x_i - \frac{f(x_i)}{f'(x_i)} \quad (6)$$

$$x_{i+1} = y_i - \frac{f(y_i)(x_i - y_i)}{f(x_i) - 2f(y_i)} \quad (7)$$

The Ostrowski method has a convergence rate resembling that of Halley's method. Both the Halley and Ostrowski methods require three function calls per iteration. This number is compared to two function calls (when using the forward or backward difference approximation to the first derivative) for the Newton method. If you us

the central difference approximation for the first derivative (which is a bit more accurate than the forward or backward difference) then each iteration in Newton's method makes three function calls. In this case, you already have the basic information that makes it easy to calculate the second derivative and graduate to using Halley's method with a small extra computational effort.

The Super Secant Algorithm

The Super Secant algorithm performs the following tasks for each iteration, given a guess for the root x for f(x)=0:

1. Let x_1=x
2. Calculate h as a small increment of x1, such as 0.01*(1 + |x_1|)
3. Calculate $x_2 = x_1 + h$
4. Calculate f(x_1) and f(x_2)
5. Estimate for the derivative $f'(x_1) = (f(x_2) - f(x_1))/(x_2 - x_1)$
6. Calculate $x_3 = x_1 - f(x_1) / f'(x_1)$
7. Calculate f(x_3) and the slope S as $(f(x_3) - f(x_1))/(x_3 - x_1)$
8. Calculate $x = x_3 - f(x_3)/S$
9. If |x_1–x| >= tolerance then resume at step 1
10. Return x as the refined root

The above steps can be summarized by the following equations:

$$x_2 = x_1 + 0.01 * (1 + |x_1|) \quad (8)$$

$$x_3 = x_1 - f(x_1) * (x_2 - x_1)/(f(x_2) - f(x_1)) \quad (9)$$

$$x_1 = x_3 - f(x_3) * (x_3 - x_1)/(f(x_3) - f(x_1)) \quad (10)$$

Let me present the pseudo-code for the new Super Secant method. Given the function f(x)=0, an initial guess, x, and a tolerance Toler for the guess:

```
Do
  LastX = X
  X1 = X
  h = 0.01 * (1 + Abs(X))
  F1 = f(X)
  X2 = X1 + h
  F2 = f(X2)
  Deriv1 = (F2 - F1) / h
  Diff = F1 / Deriv1
  X3 = X1 - Diff
  F3 = f(X3)
```

```
    If X3 <> X1 And F1 <> F3 Then
      Deriv1 = (F3 - F1) / (X3 - X1)
      Diff = F3 / Deriv1
      X = X3 - Diff
    Else
      X = X1
      LastX = X
    End If
  Loop Until Abs(LastX - X) < Toler
  Return X as the refined guess for the root.
```

The Hyper Secant Algorithm

The Hyper Secant algorithm performs the following tasks for each iteration, given a guess for the root x for f(x)=0:

1. Let $x_1=x$
2. Calculate h as a *medium-sized* increment of x_1, such as $0.1*(1+|x_1|)$
3. Calculate $x_2 = x_1 + h$
4. Calculate $f(x_1)$ and $f(x_2)$
5. Estimate for the derivative $f'(x_1) = (f(x_2) – f(x_1)) / (x_2 – x_1)$
6. Calculate $x_3 = x_1 – f(x_1) / f'(x_1)$
7. Calculate $f(x_3)$
8. Calculate $t_1 = x_1 * (0 – f(x_2) * (0 – f(x_3)) / (f(x_1) – f(x_2)) / (f(x_1) – f(x_3))$
9. Calculate $t_2 = x_2 * (0 – f(x_1)) * (0 – f(x_3)) / (f(x_2) – f(x_1)) / (f(x_2) – f(x_3))$
10. Calculate $t_3 = x_3 * (0 – f(x_1)) * (0 – f(x_2)) / (f(x_3) – f(x_1)) / (f(x_3) – f(x_2))$
11. Calculate $x = t_1 + t_2 + t_3$
12. If $|x_1–x| >=$ tolerance then resume at step 1
13. Return x as the refined root

Steps 8 through 10 performs an inverse Lagrangian interpolation to calculate x for f(x)=0, using the points $(x_1, f(x_1))$, $(x_2, f(x_2))$, and $(x_3, f(x_3))$. Thus, the Hyper Secant method combines the basic/legacy secant method and the inverse Lagrangian interpolation.

The above steps can be summarized by the following equations:

$$x_2 = x_1 + 0.1 * (1 + |x_1|) \quad (11)$$

$$x_3 = x_1 - f(x_1) * (x_2 - x_1) / (f(x_2) - f(x_1)) \quad (12)$$

$$t_1 = x_1 * (0 - f(x_2)) * (0 - f(x_3)) / (f(x_1) - f(x_2)) / (f(x_1) - f(x_3)) \quad (13)$$

$$t_2 = x_2 * (0 - f(x_1)) * (0 - f(x_3)) / (f(x_2) - f(x_1)) / (f(x_2) - f(x_3)) \quad (14)$$

$$t_3 = x_3 * (0 - f(x_1)) * (0 - f(x_2)) / (f(x_3) - f(x_1)) / (f(x_3) - f(x_2)) \quad (15)$$

$$x_1 = t_1 + t_2 + t_3 \quad (16)$$

Let me present the pseudo-code for the new Hyper Secant method. Given the function f(x)=0, an initial guess, x, and a tolerance Toler for the guess:

```
Do
  LastX = X
  X1 = X
  h = 0.1 * (1 + Abs(X1))
  F1 = f(X1)
  X2 = X1 + h
  F2 = f(X2)
  Deriv1 = (F2 - F1) / (X2 - X1)
  Diff = F1 / Deriv1
  X3 = X1 - Diff
  F3 = f(X3)
  If F1 <> F2 And F2 <> F3 And F1 <> F3 Then
    T1 = X1 * (0 - F2) * (0 - F3) / (F1 - F2) / (F1 - F3)
    T2 = X2 * (0 - F1) * (0 - F3) / (F2 - F1) / (F2 - F3)
    T3 = X3 * (0 - F1) * (0 - F2) / (F3 - F1) / (F3 - F2)
    X = T1 + T2 + t3
  Else
    X = X3
    LastX = X
  End If
Loop Until Abs(LastX - X) < Toler
Return X as the refined guess for the root.
```

Excel VBA Code

I present Excel VBA code that calculates roots using the methods of Newton, Halley, Ostrowski, and the new Super and Hyper Secant algorithms. Figure 1 shows a sample worksheet. You can download the Excel file that contains all the VBA code and the worksheets for the various tested functions.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	X	Newton		Halley		Ostrowksi		Super Secant			Hyper Secant		
2	1	0.915001	-0.0149	0.9103	-0.000870597	0.90997263	0.000104	1	0.910253	-0.00073	1	0.91018	-0.00051
3	Toler	0.910077	-0.00021	0.910008	4.41757E-08	0.910007577	-1.3E-08	0.910253	0.910008	-1.2E-09	0.91018	0.910008	-1.1E-09
4	1.00E-09	0.910008	-2.3E-06	0.910008	-2.24043E-12	0.910007572	1.65E-12	0.910008	0.910008	0	0.910008	0.910008	0
5	Fx	0.910008	-2.6E-08	0.910008	0	0.910007572	0						
6	exp(x)-3*x^2	0.910008	-2.8E-10						Fx Calls=	9		Fx Calls=	9
7		0.910008	-3.2E-12	Fx Calls=	12	Fx Calls=	12						
8													
9		Fx Calls=	12										

Figure 1. Sample Worksheet.

Note the following cells and columns in Figure 1:

- Cell A2 has the initial guess for the root.
- Cell A4 has the tolerance value.
- Cell A6 has the expression for f(x).
- Columns B and C show the output for the refined root values and their function values for Newton's method. The bottommost items in these two columns display the number of function calls for Newton's method.
- Columns D and E show the output for the refined root values and their function values for Halley's method. The bottommost items in these two the number of function calls for Halley's method.
- Columns F and G show the output for the refined root values and their function values for Ostrowski's method. The bottommost items in these two columns display the number of function calls for Ostrowski's method.
- Columns H, I, and J show the output for intermediate refined root values, the refined root values, and their function values for the new Super Secant method. The bottommost items in these two columns display the number of function calls for the Super Secant method.
- Columns K, L, and M show the output for intermediate refined root values, the refined root values, and their function values for the new Hyper Secant method. The bottommost items in these two columns display the number of function calls for the Hyper Secant method.

Here is the VBA code listing:

```
Option Explicit

Function Fx(ByVal sFx As String, ByVal X As Double) As Double
  sFx = Replace(sFx, "EXP(", "!")
  sFx = Replace(sFx, "X", "(" & X & ")")
  sFx = Replace(sFx, "!", "EXP(")
  Fx = Evaluate(sFx)
End Function

Sub doAll()
  Dim I As Integer, N As Integer

  If MsgBox("Recalculate roots in ALL worksheets containing name
'Roots'?", vbQuestion + vbYesNo, "Query") = vbNo Then
    Exit Sub
  End If

  N = Worksheets.Count
  For I = 1 To N
```

```
    'MsgBox Sheets(I).Name
    If InStr(Sheets(I).Name, "Roots") > 0 Then
      Sheets(I).Select
      Call Go
    End If
  Next I
End Sub

Sub Go()
  Dim R As Long, C As Double
  Dim X As Double, h As Double, Diff As Double
  Dim X1 As Double, X2 As Double, X3 As Double
  Dim F1 As Double, F2 As Double, F3 As Double
  Dim F0 As Double, Deriv1 As Double, Deriv2 As Double
  Dim Fp As Double, Fm As Double, LastX As Double,
  Dim Toler As Double, Z As Double, Fz As Double
  Dim sFx As String

  X = [A2].Value
  Toler = [A4].Value
  sFx = [A6].Value
  sFx = UCase(Replace(sFx, " ", ""))

  Range("B2:z1000").Clear
   On Error GoTo HandleErr

  ' Newton's method
  R = 2
  C = 2
  Do
    h = 0.01 * (1 + Abs(X))
    F0 = Fx(sFx, X)
    Diff = h * F0 / (Fx(sFx, X + h) - F0)
    X = X - Diff
    Cells(R, C) = X
    Cells(R, C + 1) = Fx(sFx, X)
    R = R + 1
  Loop Until Abs(Diff) < Toler Or R > 1000
  Cells(R + 1, C) = "Fx Calls="
  Cells(R + 1, C + 1) = 2 * (R - 2)

  ' Halley
  R = 2
  C = C + 2
  X = [A2].Value
  Do
    h = 0.01 * (1 + Abs(X))
```

```
  F0 = Fx(sFx, X)
  Fp = Fx(sFx, X + h)
  Fm = Fx(sFx, X - h)
  Deriv1 = (Fp - Fm) / 2 / h
  Deriv2 = (Fp - 2 * F0 + Fm) / h / h
  Diff = F0 / Deriv1 / (1 - F0 * Deriv2 / Deriv1 / 2 / Deriv1)
  X = X - Diff
  Cells(R, C) = X
  Cells(R, C + 1) = Fx(sFx, X)
  R = R + 1
Loop Until Abs(Diff) < Toler
Cells(R + 1, C) = "Fx Calls="
Cells(R + 1, C + 1) = 3 * (R - 2)

' Ostrowski
R = 2
C = C + 2
X = [A2].Value
Do
  LastX = X
  h = 0.01 * (1 + Abs(X))
  F0 = Fx(sFx, X)
  Fp = Fx(sFx, X + h)
  Deriv1 = (Fp - F0) / h
  Z = X - F0 / Deriv1
  Fz = Fx(sFx, Z)
  X = Z - Fz * (X - Z) / (F0 - 2 * Fz)
  Cells(R, C) = X
  Cells(R, C + 1) = Fx(sFx, X)
  R = R + 1
Loop Until Abs(X - LastX) < Toler Or R > 1000
Cells(R + 1, C) = "Fx Calls="
Cells(R + 1, C + 1) = 3 * (R - 2)


' Super-Secant
R = 2
C = C + 2
X = [A2].Value
Do
  LastX = X
  X1 = X
  h = 0.01 * (1 + Abs(X))
  F1 = Fx(sFx, X)
  X2 = X1 + h
  F2 = Fx(sFx, X2)
  Deriv1 = (F2 - F1) / h
```

```
  Diff = F1 / Deriv1
  X3 = X1 - Diff
  F3 = Fx(sFx, X3)
  If X3 <> X1 And F1 <> F3 Then
    Deriv1 = (F3 - F1) / (X3 - X1)
    Diff = F3 / Deriv1
    X = X3 - Diff
  Else
    X = X3
    LastX = X
  End If
  Cells(R, C) = X1
  Cells(R, C + 1) = X
  Cells(R, C + 2) = Fx(sFx, X)
  R = R + 1
Loop Until Abs(LastX - X) < Toler Or R > 1000
Cells(R + 1, C + 1) = "Fx Calls="
Cells(R + 1, C + 2) = 3 * (R - 2)

' Hyper-Secant
R = 2
C = C + 3
X = [A2].Value
Do
  LastX = X
  X1 = X
  h = 0.1 * (1 + Abs(X1))
  F1 = Fx(sFx, X1)
  X2 = X1 + h
  F2 = Fx(sFx, X2)
  Deriv1 = (F2 - F1) / (X2 - X1)
  Diff = F1 / Deriv1
  X3 = X1 - Diff
  F3 = Fx(sFx, X3)
  If F1 <> F2 And F2 <> F3 And F1 <> F3 Then
    X = X1 * (0 - F2) * (0 - F3) / (F1 - F2) / (F1 - F3) + _
        X2 * (0 - F1) * (0 - F3) / (F2 - F1) / (F2 - F3) + _
        X3 * (0 - F1) * (0 - F2) / (F3 - F1) / (F3 - F2)
  Else
    X = X3
    LastX = X
  End If
  Cells(R, C) = X1
  Cells(R, C + 1) = X
  Cells(R, C + 2) = Fx(sFx, X)
  R = R + 1
Loop Until Abs(LastX - X) < Toler Or R > 1000
```

```
  Cells(R + 1, C + 1) = "Fx Calls="
  Cells(R + 1, C + 2) = 3 * (R - 2)

ExitProc:
  Exit Sub
End Sub
```

Testing and Comparing the Algorithms

Table 1 shows the list of test functions. The first two functions are ones that I have chosen. The remaining functions come from the Table II in the article by Galdino, Sérgio (2011). "A family of regula falsi root-finding methods". Proceedings of 2011 World Congress on Engineering and Technology. 1. Retrieved 9 September 2016. I am using the same function numbers in Table 1 as in Table II in the article by Sérgio. I skipped functions 16 and 17 in Table II. I would like to point out that Table II, in the article by Sérgio, erroneously replicates function number 16 and 17 as function number 19 and 20, respectively. Table 1 shows the corrected form of function number 19 and 20, which are variants of function number 18.

Function Number	F(x)=
Custom 1	sin(x-1)/(x-1)-1
Custom 2	exp(x)-3*x^2
2	x^2*(x^2/3+sqrt(2)*sin(x))-sqrt(3/18)
3	11*x^11-1
4	x^3+1
5	x^3-3*x-5
6	2*x*exp(-5)+1-2*exp(-5*x)
7	2*x*exp(-10)+1-2*exp(-10*x)
8	2*x*exp(-20)+1-2*exp(-20*x)
9	(1+(1-5)^2)*x^2-(1-5*x)^2
10	(1+(1-10)^2)*x^2-(1-10*x)^2
11	(1+(1-20)^2)*x^2-(1-20*x)^2
12	x^2-(1-x)^5
13	x^2-(1-x)^10
14	x^2-(1-x)^20
15	(1+(1-5)^4)*x-(1-5*x)^4
18	exp(-5*x)*(x-1)+x^5
19	exp(-10*x)*(x-1)+x^10
20	exp(-20*x)*(x-1)+x^20
21	x^2+sin(x/5)-0.25

Function Number	F(x)=
22	x^2+sin(x/10)-0.25
23	x^2+sin(x/20)-0.25

Table 1. List of test functions.

Table 2 shows the results that compare the efficiency of the various algorithms. The comma-delimited results report the number of iterations and the number of function calls. Remember that the Newton, Halley, Ostrowski, and the two new algorithms use 2, 3, 3, 3, and 3 function calls, per iterations, respectively. The table shows one, two, and three different guesses for various test functions. The tolerance value for all the calculations is 1E-9.

Function Number	***Initial Guess***	***Newton***	***Halley***	***Ostrowski***	***Super Secant***	***Hyper Secant***
Custom 1	0	Failed	16, 48	Failed	20, 60	17, 51
Custom 2	3	13, 26	6, 18	6, 18	7, 21	6, 18
Custom 2	5	10, 20	6, 18	6, 18	5, 15	6, 18
Custom 2	-1	7, 14	4, 12	4, 12	4, 12	4, 12
Custom 2	1	6, 12	4, 12	4, 12	3, 9	3, 9
2	1	8 , 16	5, 15	5, 15	5, 15	5, 15
3	1	12, 24	7, 21	7, 21	6, 18	7, 21
4	-1.8	8, 16	5, 15	5, 15	5, 15	5, 15
5	3	8, 16	5, 15	5, 15	4, 12	4, 12
6	0	8, 16	5, 15	5, 15	4, 12	4, 12
6	1	59, 118	6, 18	18, 54	failed	22, 66
7	0	8, 16	5, 15	5, 15	4, 12	4, 12
8	0`	8, 16	5, 15	5, 15	4, 12	6, 18
9	0	6, 12	3, 9	4, 12	4, 12	3, 9
9	1	7, 14	4, 12	4, 12	4, 12	4, 12
10	0	6, 12	3, 9	4, 12	3, 9	3, 9
10	1	6, 12	3, 9	4, 12	3, 9	3, 9
11	0	5, 10	3, 9	3, 9	3, 9	3, 9
11	1	6, 12	3, 9	4, 12	3, 9	3, 9
12	0	8, 16	5, 15	4, 12	5, 15	4, 12
12	1	6, 12	5, 15	5, 15	3, 12	3, 12
13	0	9, 18	5, 15	5, 15	5, 15	4, 12
13	1	9, 18	6, 18	5, 15	5, 15	5, 15
14	0	11, 22	7, 21	6, 18	6, 18	4, 12
14	1	11, 22	7, 21	6, 18	5, 15	5, 15

Function Number	*Initial Guess*	*Newton*	*Halley*	*Ostrowski*	*Super Secant*	*Hyper Secant*
15	0	4, 8	3, 9	3, 9	3, 9	3, 9
15	1	6, 12	3, 9	4, 12	3, 9	3, 9
18	0	9, 18	6, 18	5, 15	5, 15	4, 12
18	1	9, 18	5, 15	5, 15	5, 15	5, 15
19	0	12, 24	8, 24	7, 21	7, 21	6, 18
19	1	14, 28	7, 21	7, 21	7, 21	9, 27
20	0	19, 38	12, 36	11, 33	10, 30	8, 24
20	1	24, 48	13, 39	8, 24	12, 36	15, 45
21	0	9, 18	5, 15	5, 15	5, 15	5, 15
21	1	8, 16	4, 12	5, 15	4, 12	4, 12
22	0	10, 20	6, 18	6, 18	6, 18	5, 15
22	1	8, 16	4, 12	5, 15	4, 12	4, 12
23	0	11, 22	6, 18	6, 18	6, 18	6, 18
23	1	8, 16	4, 12	5, 15	4, 12	4, 12

Table 2. Test functions for different algorithms showing the initial guesses, and the number of iterations and total function calls for each algorithm.

Looking at Table 2, you can see that the new Super & Hyper Secant algorithm do well. I use red fonts to indicate the minimum number of function calls and the minimum number of iterations. The new algorithms perform well compared to Newton's method and do slightly better than Halley and Ostrowski's methods. Thus, you have two new algorithms that you can add in your root-seeking toolbox.

Note

If you download the ZIP file containing the Excel file that contains the test functions, you can use either Excel file in the following ways:

1. Single-sheet mode. Select (or even create a copy of) a worksheet for a function you want to test. Optionally update all or some of the input parameters in cells A2, A4, and/or A6. Execute the macro **Go()** to perform the calculations on the tested root-seeking algorithms.
2. Multiple-sheets mode. You can optionally update all or some the input parameters in cells A2, A4, and/or A6, in all or some of the worksheets. To recalculate the roots in ALL of the worksheets execute macro **doAll()**. This macro will display a prompt message asking you to verify if you wish to recalculate the roots in ALL of the worksheets. Click the **Yes** button to proceed or click the **No** button to exit. The macro will quickly visit each

worksheet containing the word “Roots” in its tab name and perform the calculations. If there are no runtime errors, this macro will perform its task very quickly.