A Modified Newton and higher orders Iteration for multiple roots.

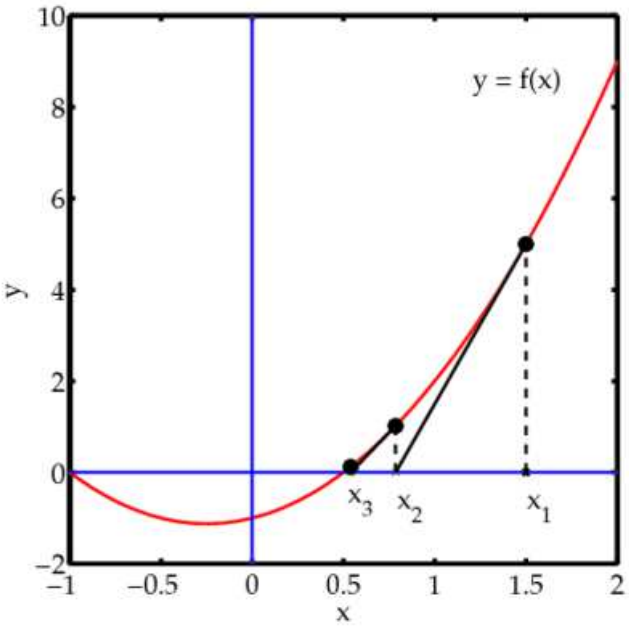
By Henrik Vestermark (hve@hvks.com)

Abstract:
In general Newton method for finding roots of polynomials is an effective and easy algorithm to both implement and use. However certain weakness is exposed when trying to find roots in a polynomial with multiple roots. This paper highlights the weakness and devised a modification to the general Newton algorithm that effectively can cope with the multiple roots issue. Furthermore we also address a solution for higher order methods as well which include Halley's and Householders 3rd order method.

Introduction:
Newton method for finding roots of polynomials is one of the most popular and simple's methods. Newton methods use the following algorithm to progressively find values closer and closer to the root.

$$x_{n+1} = x_n - \frac{P(x_n)}{P'(x_n)}$$

Graphical the next iteration step can be visualized by the interception of the tangent and the x-axis as pictured below.



Consider the polynomial:

$$P(x) = (x-1)(x-2)(x-3)(x-4) \quad or \; P(x) = x^4 - 10x^3 + 35x^2 - 50x + 24$$

Using a starting point of 0.5 the Newton iteration progress as follows towards the first root:

	x	P(x)
Initial guess	0.5	
1	0.798295454545455	6.6E+00
2	0.950817599863883	1.7E+00
3	0.996063283034122	3.2E-01
4	0.999971872651986	2.4E-02
5	0.999999998549667	1.7E-04
6	0.999999999999999	8.7E-09
7	1.000000000000000	7.1E-15

As we can see we get the first root x=1 after only 7 iterations. We also notice that after the second iteration x_2=0.95 we roughly double the number of correct digits towards the first root for each iteration. An iteration method that doubles the number of correct digits for each iteration is said to have convergence rate of 2 or quadratic convergence rate.

Now let's change the polynomial and introduce a double root at x=1;

$$P(x) = (x-1)^2(x-3)(x-4) \quad or \; P(x) = x^4 - 9x^3 + 27x^2 - 31x + 12$$

With the same starting point x=0.5 we get a much slower convergence and after 27 iterations we get no more improvement towards the first root of x=1 and the result is only accurate to approximately the first 8 digits.

	x	P(x)
Initial guess	0.5	
1	0.713414634146341	2.2E+00
2	0.842942878437970	6.2E-01
3	0.916937117337937	1.7E-01
4	0.957125910632703	4.4E-02
5	0.978193460613943	1.1E-02
6	0.988999465124113	2.9E-03
7	0.994474755305804	7.3E-04
8	0.997231047313269	1.8E-04
9	0.998613930094809	4.6E-05
10	0.999306565270595	1.2E-05
11	0.999653182516675	2.9E-06
12	0.999826566206109	7.2E-07
13	0.999913276837495	1.8E-07

14	0.999956636851837	4.5E-08
15	0.999978318031447	1.1E-08
16	0.999989158927746	2.8E-09
17	0.999994579429255	7.1E-10
18	0.999997289653755	1.8E-10
19	0.999998644904010	4.4E-11
20	0.999999322514237	1.1E-11
21	0.999999661405383	2.8E-12
22	0.999999831034522	6.9E-13
23	0.999999916015677	1.7E-13
24	0.999999956555190	4.1E-14
25	0.999999976999021	1.1E-14
26	0.999999996306426	5.3E-15
27	0.999999996306426	0.0E+00

What exactly happen here?

If $P(x) = (x-1)^2(x-3)(x-4)$ Then $P'(x) = (x-1)(4x^2 - 23x + 31)$

The root x=1 is both a root for the original Polynomial P(x) but also of P'(x). In a Newton iteration both P(x) and P'(x) go towards 0 introducing round off errors in the accuracy of calculating the next x_{n+1} in a Newton iteration. For illustration we repeat the iteration step but include the P'(x). Furthermore we introduce the convergence rate *q* as well.

	x	P(x)	P'(x)	q
Initial guess	0.5			
1	0.713414634146341	2.2E+00	-1.0E+01	
2	0.842942878437970	6.2E-01	-4.8E+00	1.3
3	0.916937117337937	1.7E-01	-2.3E+00	1.2
4	0.957125910632703	4.4E-02	-1.1E+00	1.2
5	0.978193460613943	1.1E-02	-5.4E-01	1.2
6	0.988999465124113	2.9E-03	-2.7E-01	1.2
7	0.994474755305804	7.3E-04	-1.3E-01	1.1
8	0.997231047313269	1.8E-04	-6.7E-02	1.1
9	0.998613930094809	4.6E-05	-3.3E-02	1.1
10	0.999306565270595	1.2E-05	-1.7E-02	1.1
11	0.999653182516675	2.9E-06	-8.3E-03	1.1
12	0.999826566206109	7.2E-07	-4.2E-03	1.1
13	0.999913276837495	1.8E-07	-2.1E-03	1.1
14	0.999956636851837	4.5E-08	-1.0E-03	1.1
15	0.999978318031447	1.1E-08	-5.2E-04	1.1
16	0.999989158927746	2.8E-09	-2.6E-04	1.1

17	0.999994579429255	7.1E-10	-1.3E-04	1.1
18	0.999997289653755	1.8E-10	-6.5E-05	1.1
19	0.999998644904010	4.4E-11	-3.3E-05	1.1
20	0.999999322514237	1.1E-11	-1.6E-05	1.0
21	0.999999661405383	2.8E-12	-8.1E-06	1.0
22	0.999999831034522	6.9E-13	-4.1E-06	1.0
23	0.999999916015677	1.7E-13	-2.0E-06	1.0
24	0.999999956555190	4.1E-14	-1.0E-06	1.0
25	0.999999976999021	1.1E-14	-5.2E-07	1.0
26	0.999999996306426	5.3E-15	-2.8E-07	-
27	0.999999996306426	0.0E+00	-4.4E-08	-

We notice a couple of things; the convergence rate *q* is much slower than for our first example; ~2 versus ~1.1. Furthermore we can see for each iteration that the root convergence with a linear factor of 2 instead of what we should expect the quadratic factor 2 from our first example.
For higher orders multiplicity of roots it gets even worse. E.g.

$$\text{If } P(x) = (x-1)^3(x-4) \text{ Then } P'(x) = (x-1)^2(4x-13)$$

After 31 iteration we get x=0.999998662746209 which is only accurate to approximately the first 5 digits.

	x	P(x)	P'(x)	q
Initial guess	0.5			
1	0.659090909090909	4.4E-01	-2.8E+00	
2	0.768989234449761	1.3E-01	-1.2E+00	1.2
…				
29	0.999995827925540	4.4E-16	-2.9E-10	1.0
30	0.999998662746209	4.4E-16	-1.6E-10	-
31	0.999998662746209	0.0E+00	-1.6E-11	-

What to do about multiple roots with the Newton iteration?:
Too see what is going on with the Newton iteration for multiple roots we first have to rewrite the Polynomial to the form:

$$P(x) = (x-a)^m P_r(x)$$

Where we have separated the multiple roots $(x-a)^m$ from the remainder polynomial $P_r(x)$ that have roots well separated from *a*.

Applying the Newton step $x_{n+1} = x_n - \frac{P(x_n)}{P'(x_n)}$ we get

$$x_{n+1} = x_n - \frac{(x-a)^m P_r(x\ )}{(x-a)^{m-1}(mP_r(x\ ) + (x-a)P_r'(x_n))} \quad \approx$$

$$x_{n+1} = x_n - \frac{(x-a)^m P_r(x_n)}{(x-a)^{m-1}(mP_r(x_n))}$$

When x is in the neighborhood of the multiple root *a*; or

$$x_{n+1} = g(x_n) = x_n - \frac{1}{m}\frac{(x-a)^m P_r(x_n)}{(x-a)^{m-1}P_r(x_n)}$$

Near the root *a* we will have that $g'(x_n) \approx$ is approximate constant. Differentiate above equation you get:

$$g'(a) = \frac{m-1}{m} = 1 - \frac{1}{m}$$

The above equation shows that our Newton step is reduced with a factor equivalent with the multiplicity of the root *m.* For m=2 the accuracy is only improve with a linear factor 2.

In order to overcome this reduction of the Newton step size we could multiply it with m so we instead using the modified Newton iteration.

$x_{n+1} = x_n - m\frac{P(x_n)}{P'(x_n)}$ When we encounter multiple roots.

Applying the modified equation we see that we quickly converge in just 5 iteration to the root with a quadratic factor q ~ 2 as for the normal use of the Newton method with an accuracy of more than 9 digits.

	x	P(x)	P'(x)	q
Initial guess	0.5			
1	0.903846153846154	1.3E+00	-6.5E+00	
2	0.994609766904563	3.1E-02	-6.9E-01	2.2
3	0.999980785567113	8.8E-05	-3.3E-02	2.1
4	0.999999999740396	1.1E-09	-1.2E-04	-
5	0.999999999740396	0.0E+00	-1.6E-09	-

Only issue is that we don't know beforehand if we are dealing with a multiple root issue or not. However we can use previous iteration step information to determine the multiplicity of the root.

Iterate towards *m* multiple roots the error is reduced with a constant factor per iteration given by:

$$e_{n+1} = e_n(1-\frac{1}{m})$$

As for or example with 2 multiple roots the factor is 0.5 which is exactly what we saw in the iteration step from 7 to 25 in our first example for two multiple roots.

To determine the multiplicity of the root we can use the above formula for two consecutive iteration steps.

$$e_{n+1} = e_n(1-\frac{1}{m}) \text{ and } e_n = e_{n-1}(1-\frac{1}{m}).$$

Subtracting the two equations we get

$$e_{n+1} - e_n = (e_n - e_{n-1})(1-\frac{1}{m}) \ .$$

Replacing e_{n+1} with $(x_{n+1}-a)$ near the root *a* and e_n with (x_n-a) we get.

$$x_{n+1} - x_n = (x_n - x_{n-1})(1-\frac{1}{m}) \quad \Rightarrow \frac{x_{n+1}-x_n}{x_n - x_{n-1}} = (1-\frac{1}{m}) \quad \Rightarrow$$

$$D_{n+1} = (1-\frac{1}{m}), \quad \textit{where } D_n = \frac{x_{n+1}-x_n}{x_n - x_{n-1}} \quad \Rightarrow$$

$$m = (\frac{1}{1-D_{n+1}})$$

With the above equation we can now calculate the multiplicity of the root as we do our Newton iteration. Using the polynomial

$P(x) = (x-1)^2(x-3)(x-4) \quad or \ P(x) = x^4 - 9x^3 + 27x^2 - 31x + 12$. We get

	x	P(x)	P'(x)	D_n	m	q
Initial guess	0.5					
1	0.713414634146341	2.2E+00	-1.0E+01			
2	0.842942878437970	6.2E-01	-4.8E+00			1.3
3	0.916937117337937	1.7E-01	-2.3E+00	0.57	2.3	1.2
4	0.957125910632703	4.4E-02	-1.1E+00	0.54	2.2	1.2
5	0.978193460613943	1.1E-02	-5.4E-01	0.52	2.1	1.2
6	0.988999465124113	2.9E-03	-2.7E-01	0.51	2.1	1.2
7	0.994474755305804	7.3E-04	-1.3E-01	0.51	2.0	1.1

8	0.997231047313269	1.8E-04	-6.7E-02	0.50	2.0	1.1
9	0.998613930094809	4.6E-05	-3.3E-02	0.50	2.0	1.1
10	0.999306565270595	1.2E-05	-1.7E-02	0.50	2.0	1.1
11	0.999653182516675	2.9E-06	-8.3E-03	0.50	2.0	1.1
12	0.999826566206109	7.2E-07	-4.2E-03	0.50	2.0	1.1
13	0.999913276837495	1.8E-07	-2.1E-03	0.50	2.0	1.1
14	0.999956636851837	4.5E-08	-1.0E-03	0.50	2.0	1.1
15	0.999978318031447	1.1E-08	-5.2E-04	0.50	2.0	1.1
16	0.999989158927746	2.8E-09	-2.6E-04	0.50	2.0	1.1
17	0.999994579429255	7.1E-10	-1.3E-04	0.50	2.0	1.1
18	0.999997289653755	1.8E-10	-6.5E-05	0.50	2.0	1.1
19	0.999998644904010	4.4E-11	-3.3E-05	0.50	2.0	1.1
20	0.999999322514237	1.1E-11	-1.6E-05	0.50	2.0	1.0
21	0.999999661405383	2.8E-12	-8.1E-06	0.50	2.0	1.0
22	0.999999831034522	6.9E-13	-4.1E-06	0.50	2.0	1.0
23	0.999999916015677	1.7E-13	-2.0E-06	0.50	2.0	1.0
24	0.999999956555190	4.1E-14	-1.0E-06	0.48	1.9	1.0
25	0.999999976999021	1.1E-14	-5.2E-07	0.50	2.0	1.0
26	0.999999996306426	5.3E-15	-2.8E-07	0.94	18.0	-
27	0.999999996306426	0.0E+00	-4.4E-08	0.00	1.0	-

Notice that D_n as expected is 0.5 and *m* is 2. For a polynomial with multiplicity of 3 we get: E.g.

$$P(x) = (x-1)^3(x-4) \quad or\ P(x) = x^4 - 7x^3 + 15x^2 - 13x + 4\,.$$

	x	P(x)	P'(x)	D_n	m	q
Initial guess	0.5					
1	0.659090909090909	4.4E-01	-2.8E+00			
2	0.768989234449761	1.3E-01	-1.2E+00			1.2
3	0.844200342036924	4.0E-02	-5.3E-01	0.68	3.2	1.1
4	0.895292762147974	1.2E-02	-2.3E-01	0.68	3.1	1.1
5	0.929807171624113	3.6E-03	-1.0E-01	0.68	3.1	1.1
6	0.953027819025432	1.1E-03	-4.6E-02	0.67	3.1	1.1
7	0.968605165792479	3.2E-04	-2.0E-02	0.67	3.0	1.1
8	0.979034107860224	9.4E-05	-9.0E-03	0.67	3.0	1.1
9	0.986006608556179	2.8E-05	-4.0E-03	0.67	3.0	1.1
10	0.990663864788857	8.3E-06	-1.8E-03	0.67	3.0	1.1
11	0.993772694924434	2.4E-06	-7.9E-04	0.67	3.0	1.1
12	0.995847030978970	7.3E-07	-3.5E-04	0.67	3.0	1.1
13	0.997230716376341	2.2E-07	-1.6E-04	0.67	3.0	1.1

14	0.998153527238367	6.4E-08	-6.9E-05	0.67	3.0	1.1
15	0.998768892029038	1.9E-08	-3.1E-05	0.67	3.0	1.1
16	0.999179205334757	5.6E-09	-1.4E-05	0.67	3.0	1.1
17	0.999452778575931	1.7E-09	-6.1E-06	0.67	3.0	1.0
18	0.999635174582289	4.9E-10	-2.7E-06	0.67	3.0	1.0
19	0.999756779088727	1.5E-10	-1.2E-06	0.67	3.0	1.0
20	0.999837850465727	4.3E-11	-5.3E-07	0.67	3.0	1.0
21	0.999891897587627	1.3E-11	-2.4E-07	0.67	3.0	1.0
22	0.999927925394456	3.8E-12	-1.1E-07	0.67	3.0	1.0
23	0.999951956279215	1.1E-12	-4.7E-08	0.67	3.0	1.0
24	0.999967946216671	3.3E-13	-2.1E-08	0.67	3.0	1.0
25	0.999978847780829	1.0E-13	-9.2E-09	0.68	3.1	1.0
26	0.999985795662453	2.8E-14	-4.0E-09	0.64	2.8	1.0
27	0.999990197716745	8.0E-15	-1.8E-09	0.63	2.7	1.0
28	0.999994305994514	3.6E-15	-8.6E-10	0.93	15.0	1.1
29	0.999995827925540	4.4E-16	-2.9E-10	0.37	1.6	1.0
30	0.999998662746209	4.4E-16	-1.6E-10	1.86	-1.2	-
31	0.999998662746209	0.0E+00	-1.6E-11	0.00	1.0	-

Again we get $D_n = 2/3$ and m = 3 as expected.

And by using the multiplier m=3 we get:

Multiplier	3					
	x	P(x)	P'(x)	Dn	m	q
Initial guess	0.5					
1	0.977272727272727	4.4E-01	-2.8E+00			
2	0.999943181817001	3.5E-05	-4.7E-03			2.6
3	1.000000084812110	5.5E-13	-2.9E-08	0.00	1.0	0.4
4	1.019736926917380	4.4E-16	-6.8E-14	346.85	0.0	1.0
5	0.999956334044330	-2.3E-05	-3.5E-03	-1.00	0.5	2.6
6	0.999999964612765	2.5E-13	-1.7E-08	0.00	1.0	-
7	0.999999964612765	0.0E+00	-1.1E-14	0.00	1.0	-

With 7 correct digits after 7 iterations. Also notice that the Newton was throwing a little bit of course in iteration 4 due to the round off errors is strong around the multiple root.

Practical implementation

I have seen one implementation of the above strategy and that is Madsen [1] that use a variable Newton step size to ensure quadratic convergence of the Newton method even for multiple roots. Other methods have been described in McNamee [2].

Halley's method

Let turn our attention to a higher order method. One of them is Halley which is a cubic convergence method meaning that for each iteration step we triple the number of correct digits in our root.

The Halley's method use the iteration:

$$x_{n+1} = x_n - \frac{2P(x_n)P'(x_n)}{2P'(x_n)^2 - P(x_n)P''(x_n)}$$

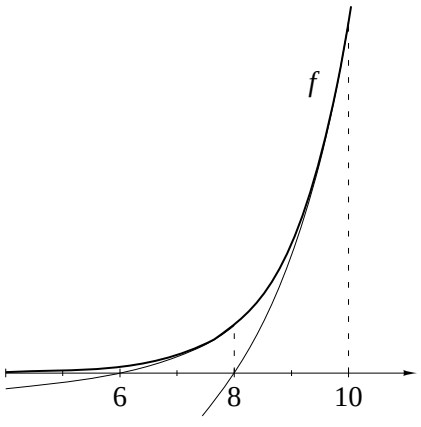
Or sometimes written as: ([5] Peter Acklam)

$$x_{n+1} = x_n - \frac{P(x_n)}{P'(x_n)}\left[1 - \frac{P(x_n)P''(x_n)}{2P'(x_n)^2}\right]^{-1}$$

Where $x_{n+1} = x_n - \frac{P(x_n)}{P'(x_n)}$ is the usual Newton iteration enhanced with the factor:

$$\left[1 - \frac{P(x_n)P''(x_n)}{2P'(x_n)^2}\right]^{-1}$$

And are graphic shows below:



Applied to our usual test polynomial:

$$P(x) = (x-1)(x-2)(x-3)(x-4) \quad or \; P(x) = x^4 - 10x^3 + 35x^2 - 50x + 24$$

We get using a starting point of 0.5 the Halley iteration progress as follows towards the first root:

	x	P(x)	m	q
Initial guess	0.5			
1	0.921033445730429	6.6E+00		
2	0.999101217617920	5.5E-01		2.8
3	0.999999998290928	5.4E-03	1.0	2.9
4	1.000000000000000	1.0E-08	1.0	1.7
5	1.000000000000000	-7.1E-15	1.0	1.0

Using 2 iteration less than the Newton method and a convergence rate that approximate the theoretical 3 for cubic methods.

Applied to our multiple root test polynomial we get as expected a much slower convergence but still significant better than the Newton methods 27 iterations versus Halley 17 iterations.

$$P(x) = (x-1)^2(x-3)(x-4) \quad or\ P(x) = x^4 - 9x^3 + 27x^2 - 31x + 12$$

	x	P(x)	m	q
Initial guess	0.5			
1	0.810337370242215	2.2E+00		
2	0.933368912312335	2.5E-01		1.5
3	0.977372635121701	2.8E-02	2.1	1.3
4	0.992409863611796	3.1E-03	2.0	1.3
5	0.997464609321869	3.5E-04	2.0	1.2
6	0.999154274154362	3.9E-05	2.0	1.2
7	0.999718025142576	4.3E-06	2.0	1.1
8	0.999906001018135	4.8E-07	2.0	1.1
9	0.999968666187712	5.3E-08	2.0	1.1
10	0.999989555309201	5.9E-09	2.0	1.1
11	0.999996518432377	6.5E-10	2.0	1.1
12	0.999998839531160	7.3E-11	2.0	1.1
13	0.999999612788777	8.1E-12	2.0	1.1
14	0.999999870639296	9.0E-13	2.0	1.1
15	0.999999957852718	1.0E-13	2.0	1.1
16	0.999999980034920	8.9E-15	1.7	-
17	0.999999980034920	0.0E+00	1.0	-

As for the Newton methods applying the same technique we find that D_n is giving by

$$D_{n+1} = 1 - \frac{2}{m+1}, \quad where\ D_n = \frac{x_{n+1} - x_n}{x_n - x_{n-1}} \quad \Rightarrow$$

$$m = \frac{2}{1 - D_{n+1}} - 1$$

Equivalent with the Newton reduction the Halley reduction is a factor of 2/(m+1) so by multiplier the step size with the reverse factor we should ensure cubic convergence rate.

Our modified Halley will be:

$$x_{n+1} = x_n - \frac{m+1}{2} \frac{2P(x_n)P'(x_n)}{2P'(x_n)^2 - P(x_n)P''(x_n)}$$

Which could also be written as:

$$x_{n+1} = x_n - \frac{m+1}{2} \frac{P(x_n)}{P'(x_n)} \left[1 - \frac{P(x_n)P''(x_n)}{2P'(x_n)^2} \right]^{-1}$$

Using our modified Halley iteration we get using our multiple root test polynomials:

$$P(x) = (x-1)^2(x-3)(x-4) \quad or\ P(x) = x^4 - 9x^3 + 27x^2 - 31x + 12$$

	x	P(x)	m	q
Initial guess	0.5			
1	0.965506055363322	2.2E+00		
2	0.999833351530136	7.3E-03		2.6
3	0.999999996140290	1.7E-07	1.0	2.1
4	0.999999986501257	-1.8E-15	1.0	0.9
5	1.000000014202640	1.8E-15	-0.5	-
6	1.000000014202640	0.0E+00	1.0	-

Again as we expected approaching cubic convergence and finding the root to 7 correct digits in just 6 iterations.

Hansen & Patrick [4] is a special case of Halley and use this variation:

$$x_{n+1} = x_n - \frac{P(x_n)}{\frac{m+1}{2m} P'(x_n) - \frac{P(x_n)P''(x_n)}{2P'(x_n)}}$$

Or written in another way:

$$x_{n+1} = x_n - \frac{P(x_n)}{P'(x_n)}\left[\frac{m+1}{2m} - \frac{P(x_n)P''(x_n)}{2P'(x_n)^2}\right]^{-1}$$

Where $x_{n+1} = x_n - \frac{P(x_n)}{P'(x_n)}$ is the usual Newton iteration modified with the factor:

$$\left[\frac{m+1}{2m} - \frac{P(x_n)P''(x_n)}{2P'(x_n)^2}\right]^{-1}$$

This could also have been shown using Halley formula:

$$x_{n+1} = x_n - \frac{P(x_n)}{P'(x_n)}\left[1 - \frac{P(x_n)P''(x_n)}{2P'(x_n)^2}\right]^{-1}$$

Replacing $P(x_n)$ with:

$$P(x) = (x-a)^m P_r(x)$$
$$P'(x) = (x-a)^{m-1}((x-a)P_r'(x) + mP_r(x))$$
$$P''(x) = (x-a)^{m-2}((x-a)^2 P_r''(x) + 2m(x-a)P_r'(x) + m(m-1)P_r(x))$$

Into : $\left[1 - \frac{P(x_n)P''(x_n)}{2P'(x_n)^2}\right]^{-1}$

You get:

$$\left[1 - \frac{P_r(x_n)\left[(a-x)^2 P_r''(x_n) + 2m(x-a)P_r'(x_n) + (m-1)mP_r(x_n)\right]}{2((x-a)P_r'(x_n) + mP_r(x_n))^2}\right]^{-1}$$

Eliminating all $(a-x)^2$ and $(x-a)$ when close to the root *a* you get

$$\left[1 - \frac{P_r(x_n)\left[(m-1)mP_r(x_n)\right]}{2(mP_r(x_n))^2}\right]^{-1} \Rightarrow \left[1 - \frac{(m-1)m}{2m^2}\right]^{-1} \Rightarrow \left[\frac{m+1}{2m}\right]^{-1} \Rightarrow \frac{2m}{m+1}$$

For m=1 we get the factor 1, m=2 we get the factor 1.33 and for m=3 we get 1.5 and for m=4 we get 1.6. Substituting this result into our original

$$x_{n+1} = x_n - \frac{m+1}{2}\frac{P(x_n)}{P'(x_n)}\frac{2m}{m+1} \Rightarrow$$

$$x_{n+1} = x_n - m\frac{P(x_n)}{P'(x_n)}$$

Which is the usual Newton iteration for multiple roots.

Using [4] for multiple roots or our modified Halley iteration we get similar results.

Householder 3rd order method

Let turn our attention to the next higher order methods. Household has generalized the higher order methods in which 1st order is the Newton and 2nd order is Halley's method. Householders 3rd order has quantic convergence rate.

The Householder's 3rd order method uses the following iteration:

$$x_{n+1} = x_n - \frac{6P(x_n)P'(x_n)^2 - 3P(x_n)^2 P''(x_n)}{6P'(x_n)^3 - 6P(x_n)P'(x_n)P''(x_n) + P(x_n)^2 P'''(x_n)}$$

See [3] Pascal Sebah and Xavier Gourdon (2001.

Substituting:

$$t = \frac{P(x_n)}{P'(x_n)}$$
$$u = \frac{P''(x_n)}{P'(x_n)}$$
$$v = \frac{P'''(x_n)}{P'(x_n)}$$

We can now write the householder's 3rd order as following:

$$x_{n+1} = x_n - \frac{t(1-0.5tu)}{1-t(u-\frac{vt}{6})}$$

Applied to our usual test polynomial:

$$P(x) = (x-1)(x-2)(x-3)(x-4) \quad or \; P(x) = x^4 - 10x^3 + 35x^2 - 50x + 24$$

We get using a starting point of 0.5 the Householder iteration progress as follows towards the first root:

	x	P(x)	m	q
Initial guess	0.5			
1	0.970345147974213	6.6E+00		
2	0.999998181755405	1.9E-01		3.8
3	1.000000000000000	1.1E-05	1.0	-
4	1.000000000000000	0.0E+00	1.0	-

Using 3 iteration less than the Newton method and one less than the Halley's method and a convergence rate that approximate the theoretical 4 for quartic methods.

Applied to our multiple root test polynomial we get as expected a much slower convergence but still significant better than the Newton methods 27 iterations versus Halley 17 iterations and Householder 3rd order of 14.

$$P(x) = (x-1)^2(x-3)(x-4) \quad or \; P(x) = x^4 - 9x^3 + 27x^2 - 31x + 12$$

	x	P(x)	m	q
Initial guess	0.5			
1	0.861059798855960	2.2E+00		
2	0.964231209357945	1.3E-01		1.6
3	0.990990668543017	7.9E-03	2.1	1.4
4	0.997743431685646	4.9E-04	2.0	1.3
5	0.999435592582946	3.1E-05	2.0	1.2
6	0.999858881551644	1.9E-06	2.0	1.2
7	0.999964719352181	1.2E-07	2.0	1.2
8	0.999991179774371	7.5E-09	2.0	1.1
9	0.999997794908371	4.7E-10	2.0	1.1
10	0.999999448583258	2.9E-11	2.0	1.1
11	0.999999862128834	1.8E-12	2.0	1.1
12	0.999999970461575	1.2E-13	2.1	1.1
13	0.999999993287772	5.3E-15	1.8	-
14	0.999999993287772	0.0E+00	1.0	-

As for the Newton methods when applying the same technique we find that D_n for householder 3rd order is giving by

$$D_{n+1} = 1 - \frac{3}{m+2}, \quad where\ D_n = \frac{x_{n+1} - x_n}{x_n - x_{n-1}} \quad \Rightarrow$$

$$m = \frac{3}{1 - D_{n+1}} - 2$$

Equivalent with the Newton reduction the Householder 3rd order reduction is a factor of 3/(m+2) so by multiplier the step size with the reverse factor we should ensure quartic convergence rate.

Our modified Householder 3rd order will be:

$$x_{n+1} = x_n - \frac{m+2}{3}\left[\frac{6P(x_n)P'(x_n)^2 - 3P(x_n)^2 P''(x_n)}{6P'(x_n)^3 - 6P(x_n)P'(x_n)P''(x_n) + P(x_n)^2 P'''(x_n)} \right]$$

Or using the same substitution as before:

$$x_{n+1} = x_n - \frac{m+2}{3}\left[\frac{t(1 - 0.5tu)}{1 - t(u - \frac{vt}{6})} \right]$$

Using our modified Householder 3rd order iteration we get using our multiple root test polynomials:

$$P(x) = (x-1)^2(x-3)(x-4) \quad or\ P(x) = x^4 - 9x^3 + 27x^2 - 31x + 12$$

	x	P(x)	m	q
Initial guess	0.5			
1	0.981413065141280	2.2E+00		
2	0.999975915594327	2.1E-03		2.7
3	0.999999999958017	3.5E-09	1.0	-
4	0.999999999958017	0.0E+00	1.0	-

Again as we expected approaching cubic convergence and finding the multiple root to 10 correct digits in just 4 iterations.

Conclusion

Newton, Halley's and higher order Householder can all be modified to deal with the multiple roots issue that will ensure the same convergence rate as for non-multiple roots. Thereby eliminating one of the weaknesses these methods has.

Reference

1. Kaj Madsen, A Root-finding Algorithm based on Newton's Method, Bit 13 (1973) page 71-75
2. McNamee, J.M., Numerical Methods for Roots of Polynomials, Part I, Elsevier, Kidlington, Oxford 2009.
3. Pascal Sebah and Xavier Gourdon (2001). "Newton's method and high order iteration"
4. E Hansen & M. Ptraick, A family of root finding methods, Numerical. Math 27 (1077) 257-269
5. Peter Acklam, A small paper on Halley's method, http://home/online.no~pjacklam, 23rd December 2002