Very Brief Comments on the Results

By

Namir Shammas

On April 3, 2017, I started a thread on www.hpmuseum.com discussing new root-seeking algorithms (see http://www.hpmuseum.org/forum/thread-8092.html). The discussion brought several external links, shared by others, for several modified Newton's methods. I visited these links and similar ones to obtain a current survey for modified Newton's methods. The ZIP file in which you found this file has an Excel file and several PDF files for the source articles that I collected.

The Excel file *Newton Modifications April 2017.xlsm* contains several spreadsheets that compares Newton's method with 58 other modifications of Newton's method. The VBA code (in the *ThisWorkbook* module) has a subroutine Go() which contains code and comments that site the reference articles and equations used and the method applied. Please note the following:

- Some articles site other methods, which I also included in the tests.
- Some articles present families of root-seeking algorithms. In this case, I have presented several examples of these algorithm instances.
- Each method sites the reference article's author, full title, and publication name and date.
- I include PDF files associated with the sited reference in the VBA comments. The names of these PDF files are made up of:
    - The last name of the lead author.
    - A leading part of the article's full title. I use part of the full title to keep the filename relatively short.

The Excel workbook contains several worksheets. Each worksheet test for a specific nonlinear function with a given initial guess and tolerance value. The output for each algorithm tested comprises of two columns—the current guess value and its related function value. At the end of the values appears the number of function calls. The macro SUB doAll() allows you to quickly recalculate the roots for all the examples using all of the root-seeking methods. If you examine the VBA code in subroutine Go() (in the *ThisWorkbook* module) you see that the code for each algorithm has its own error handler. Such error handlers allow the code execution to resume to the

next algorithm if a run-time error occurs in the current algorithm. When a runtime error occurs, you will see that the tabulated results of the offending algorithm, in a worksheet, has NO function calls count! Displaying the current number of function calls as a *regular result* under runtime error would be misleading!

The *Summary* worksheet displays a summary for all the algorithms and for all the worksheets (except *CustomRoots1* which is very difficult to solve). The *Summary* worksheet tabulates the number of iterations and function calls for each algorithm. If an algorithm experiences a runtime error, the VBA code inserts the value of 1 million as the number of function calls

Rows 39 and 40 contain the mean number of iterations and the mean function calls for each algorithm based on the examples. Row 41 calculates special weight factors that combine the number of iterations and the function calls using:

Wt = wt*iterations + fx_Calls

The value of wt appears in cell B42 (current set to 0.65). The above formula shows that I put more weight to the number of function calls than the number of iterations. The weight factor makes it easier to rank values that combine both the number of iterations and function calls. You can alter the value for the weight in cell B42 value. This change allows you to see how the results of the ranks in rows 44, 45, and 46 change.

The VBA code associated with the *Summary* worksheet contains SUB GetSummary() that will populate rows 2 to 36 in the worksheet *Summary*, using the results in all of the other worksheets, except *CustomRoots*. Use this subroutine to update the data in rows 2 to 36. This allows you to update the entries for different initial guess, tolerance values, and even test different equations. If you add more worksheets to test additional functions, you need to insert blank rows (using Excel's Insert command) to push down rows 39 and below. You may also need to update the cells ranges used in the average-calculating formulas in rows 39 and 40 to cover the additional rows. I strongly suggest that you make a copy of the Excel file and experiment with that copy.

Rows 44, 45, and 46 rank the values of the mean iterations, mean function calls, and weight factors. I will focus on the results of the weight factors next.

The best weight factor (which combines the best values for the number of iterations and function calls) belongs to the Grau algorithm.

The second-best weight factor belongs to the Potra-Ptak algorithm. The third best weight factor belongs to Rahimi 1 algorithm.

In general, the various modified Newton algorithms reach their solutions in fewer iterations that Newton's method. However, this advantage comes usually at a price of high number of function calls. You will find several algorithms that do better that Newton in both the number of iterations AND function calls.

Another interesting aspect is that most of the modified Newton algorithms use one or more intermediate refined guesses for the root per iteration. This approach is inspired by Ostrowski's modification for Newton's method.