GRAPHIC AND NUMERICAL COMPARISON BETWEEN ITERATIVE METHODS

JUAN L. VARONA

Dedicated to the memory of José J. Guadalupe "Chicho", my Ph. D. Advisor.

Introduction

Let f be a function $f\colon \mathbb{R} \to \mathbb{R}$ and ζ a root of f, that is, $f(\zeta) = 0$. It is well known that if we take x_0 close to ζ, and under certain conditions that I will not explain here, the Newton method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \dots$$

generates a sequence $\{x_n\}_{n=0}^{\infty}$ that converges to ζ. In fact, Newton's original ideas on the subject, around 1669, were considerably more complicated. A systematic study and a simplified version of the method are due to Raphson in 1690, so this iteration scheme is also known as the Newton-Raphson method. (Also as the tangent method, from its geometric interpretation.)

In 1879, Cayley tried to use the method to find complex roots of complex functions $f\colon \mathbb{C} \to \mathbb{C}$. If we take $z_0 \in \mathbb{C}$ and we iterate

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}, \quad n = 0, 1, 2, \dots, \tag{1}$$

he looked for conditions under which the sequence $\{z_n\}_{n=0}^{\infty}$ converges to a root. In particular, if we denominate the *attraction basin* of a root ζ as the set of all $z_0 \in \mathbb{C}$ such that the method converges to ζ, he was interested in identifying the attraction basin for any root. He solved the problem when f is a quadratic polynomial. For cubic polynomials, after several years of trying, he finally refused to continue. We now know the fractal nature of the problem and we can understand that Cayley's failure to make any real progress at that time was inevitable. For instance, for $f(z) = z^3 - 1$, the Julia set (that is, the points where Newton's method fails to converge) has fractional dimension, and it coincides with the frontier of the attraction basins of the three complex roots $e^{2k\pi i/3}$, $k = 0, 1, 2$. With the aid of computer generated pictures, we can see nice figures that show the complexity of these intricate regions. In Figure 1, I show the attraction basins of the three roots (actually, this picture is well known; for instance, it already appears published in [5] and, later, [16] and [21]).

The computational aspect of iterative methods is focused in two ways: (a) to actually find roots of nonlinear equations, studying also the accuracy and stability of the numerical algorithms; (b) to show the beauty of the pictures that can be generated with the aid of computers. The first point of view is covered by numerical analysis. General books on this subject are [9, 13]; and more specialized books

Date: First version: February 4, 1999. Present version: April 11, 2001.

This paper has been published, with small changes, in *The Mathematical Intelligencer* **24** (2002), no. 1, 37–46. Also, the cover of the issue has been designed by using pictures from this paper (Figures 3, 8, 9, and 12).

on iterative methods are [3, 15, 18]. For the graphical point of view, see, for instance, [16].

Generally, there are three strategies to obtain pictures by using Newton's method:

(i) We take a rectangle $D \subset \mathbb{C}$ and we assign a color (or a gray level) to each point $z_0 \in D$ according to the root at which Newton's method starting from z_0 converges, and we mark the point as black (for instance) if the method does not converge. In this way, we distinguish the attraction basins by their colors.

(ii) Instead of assigning the color according to the root reached by the method, we assign the color according to the number of iterations required to reach some root with a fixed precision. Again, black is used if the method does not converge. This lacks mathematical interpretation as regards the Julia sets, but it also generates nice pictures.

(iii) This is a combination of the two previous strategies. Here, we assign a color to each attraction basin of a root. But we make the color lighter or darker according to the number of iterations needed to reach the root with the fixed precision required. As before, we use black if the method does not converge. In my opinion, this generates the most beautiful pictures.

All these strategies have been extensively used for polynomials, mainly for polynomials with the form $z^n - 1$ whose roots are well known. Of course, many other families of functions have been studied. We refer the reader to [4, § 6] to find further references. For instance, a nice picture appears when we apply the method to the polynomial $(z^2-1)(z^2+0.16)$ (due to S. Sutherland, see [17, the cover of issue no. 2 for a color version]).

Although Newton's method is the best known, in the literature there are many other iterative methods devoted to finding roots of nonlinear equations. Thus, the aim of this paper is the following: using computer experiments, we study some of these iterative methods for solving $f(z) = 0$, where $f\colon \mathbb{C} \to \mathbb{C}$, and we show the fractal pictures that they generate (mainly, in the sense described in (iii)). Also, this allows us to compare the regions of convergence of the methods and their speed.

Concepts related with the speed of convergence

Let $\{z_n\}_{n=0}^{\infty}$ be a complex sequence. We say that $\alpha \in [1,\infty)$ is the *order of convergence* of the sequence if

$$\lim_{n\to\infty} \frac{|z_{n+1}-\zeta|}{|z_n-\zeta|^{\alpha}} = C, \tag{2}$$

where ζ is a complex number and C a nonzero constant; here, if $\alpha = 1$, we assume an extra condition $C < 1$. Then, the convergence of order α implies that the sequence $\{z_n\}_{n=0}^{\infty}$ converges to ζ when $n \to \infty$. (The previous definition for the order of convergence can be extended under some circumstances; but I will not worry about that.) Also, it is said that the order of convergence is at least α if the constant C in (2) is allowed to be 0, or, the equivalent, if there exists a constant C and an index n_0 such that $|z_{n+1}-\zeta| \le C|z_n-\zeta|^{\alpha}$ for any $n \ge n_0$. Many times, the *at least* is supposed implicitly. I will do so in this paper.

The order of convergence is used to compare the speed of convergence of sequences, understanding the speed as the number of iterations necessary to reach the limit with a required precision. Suppose that we have two sequences $\{z_n\}_{n=0}^{\infty}$ and $\{z'_n\}_{n=0}^{\infty}$ converging to the same limit ζ, and assume that they have, respectively, orders of convergence α and α', where $\alpha > \alpha'$. Then, it is clear that, asymptotically, the sequence $\{z_n\}_{n=0}^{\infty}$ converges to the limit more quickly (with less iterations for the same approximation) than the other sequence.

More refined measures for the speed of convergence are the concepts of *informational efficiency* and *efficiency index* (see [18, § 1.24]). If each iteration requires d new pieces of information (a "piece of information" typically is any evaluation of a function or one of its derivatives), then the informational efficiency is $\frac{\alpha}{d}$ and the efficiency index is $\alpha^{1/d}$, where α is the order of convergence. For the methods that I are dealing with in this paper, it is easy to derive both the informational efficiency and the efficiency index from its order. I will do this here for the efficiency index.

The efficiency index is useful because it allows us to avoid artificial accelerations of an iterative method: for instance, let us suppose that we have an iterative process $z_{n+1} = \phi(z_n)$ with order of convergence α and we take a new process $z_0^* = z_0$, $z_{n+1}^* = \phi(\phi(z_n^*))$. Then, it is clear that the new sequence is merely $z_n^* = z_{2n}$, but $\{z_n^*\}_{n=0}^\infty$ has order of convergence α^2. However, both sequences $\{z_n\}_{n=0}^\infty$ and $\{z_n^*\}_{n=0}^\infty$ have the same efficiency index.

In my opinion, when we have an iterative method $z_{n+1} = \phi(z_n)$, the efficiency index is more suitable than the order of convergence to actually measure the computer time that a method uses to converge. But, as happens in our case, if ϕ is related with a function f and its derivatives, the efficiency index still has a missing element: it does not take into account the computational work involved in computing $f, f', \dots$. To avoid this, a new concept of efficiency is given: the *computational efficiency* (see [18, Appendix C]). Suppose that, in a method ϕ related with a function f, the cost of evaluating ϕ is $\theta(f)$ (for instance, in Newton's method, if the cost of evaluating f and f' are, respectively, θ_0 and θ_1, we have $\theta(f) = \theta_0 + \theta_1$); then, the computational efficiency of ϕ relative to f is $E(\phi, f) = \alpha^{1/\theta(f)}$ where, again, α is the order of convergence. But it is difficult to clearly establish the value of $\theta(f)$ and, moreover, it can depend on the computer, so the computational efficiency is not very much used in practice. Thus, in the literature, the most used of these measures is the order of convergence; however, this is the one that provides least information about the computer time necessary to find the root with a required precision.

Finally, let us note that, to ensure the convergence of an iterative method $z_{n+1} = \phi(z_n)$ intended for solving an equation $f(z) = 0$, it is usually necessary to begin the method from a point z_0 close to the solution ζ. How close depends on ϕ and f. Usually, the hypothesis of the theorems that guarantee the convergence (I refer the reader to the references of each method) are hard to check; and, moreover, are too demanding. So, if we want to solve $f(z) = 0$, it is common to try a method without take into account any hypothesis. Of course, this does not guarantee convergence, but it is possible that we will find a solution (if there is more than one solution, we also cannot know which solution is going to be found).

Here, we will do some numerical experiments with different functions (simple and hard to evaluate) that allow us to compare the computational time used. In addition, we will begin the iterations in different regions of the complex plane. This will allow us to measure to some extent how demanding the method is regarding the starting point to find a solution. As the fractal that appears becomes more complicated, it seems that the method requires more conditions on the initial point.

The numerical methods

In this section, let us consider some iterative methods $z_{n+1} = \phi(z_n)$ for solving $f(z) = 0$ for a complex function $f\colon \mathbb{C} \to \mathbb{C}$. I only give a brief description and a few references for them. In all these methods, we take a starting point $z_0 \in \mathbb{C}$. In no case do I give the hypothesis that ensures convergence and the corresponding order of convergence.

- Newton's method: This is the iterative method (1), the best known and most used, and can be found in any book on numerical analysis. I have already commented on it in the introduction. Its order of convergence is 2.
- Newton's method for multiple roots:
$$z_{n+1} = z_n - \frac{f(z_n)f'(z_n)}{f'(z_n)^2 - f(z_n)f''(z_n)}.$$
Actually, Newton's method has order 2 when the root of f that is found is a simple root. For a multiple root, its order of convergence is 1. This method recovers the order 2 for multiple roots. It can be deduced as follows: if f has a root of multiplicity $m \geq 1$ in ζ, it is easy to check that $g(z) = \frac{f(z)}{f'(z)}$ has a simple root in ζ. Then, we only need to apply the ordinary Newton's method to the equation $g(z) = 0$.
- Convex acceleration of Whittaker's method [11]:
$$z_{n+1} = z_n - \frac{f(z_n)}{2f'(z_n)}(2 - L_f(z_n))$$
with
$$L_f(z) = \frac{f(z)f''(z)}{f'(z)^2}.$$
Whittaker's method (also known as the parallel-chord method, from its geometric interpretation for functions $f\colon \mathbb{R} \to \mathbb{R}$, see [15, p. 181]) is a simplification of Newton's method in which, to avoid computing the derivative, we make the approximation $f'(z) \approx 1/\lambda$ with λ a constant. We try to choose the parameter λ in such a way that $F(z) = z - \lambda f(z)$ is a contractive function, so a fixed point can be obtained by means of the fixed point theorem (it is clear that a fixed point for F is a root for f). This is a method of order 1. The convex acceleration is an order 2 method.
- Double convex acceleration of Whittaker's method [11]:
$$z_{n+1} = z_n - \frac{f(z_n)}{4f'(z_n)}\left(2 - L_f(z_n) + \frac{4 + 2L_f(z_n)}{2 - L_f(z_n)(2 - L_f(z_n))}\right).$$
This is a new convex acceleration for the previous iterative process. It has order 3.
- Halley's method (see [18, p. 91], [3, p. 247], [9, p. 257], [8]):
$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}\frac{2}{2 - L_f(z_n)} = z_n - \frac{1}{\frac{f'(z_n)}{f(z_n)} - \frac{f''(z_n)}{2f'(z_n)}}.$$
This was presented in about 1694 by Edmund Halley, who is well known for first computing the orbit of the Halley comet. It is one of the most frequently rediscovered iteration functions in the literature. From its geometric interpretation for real functions, it is also known as the method of tangent hyperbolas. Alternatively, it can be interpreted as applying Newton's method to the equation $g(z) = 0$ with $g(z) = f(z)/\sqrt{f'(z)}$. Its order of convergence is 3.
- Chebyshev's method (see [18, p. 76 and p. 81] or [3, p. 246]):
$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}\left(1 + \frac{L_f(z_n)}{2}\right).$$
This is also know as Euler-Chebyshev's method or, from its geometric interpretation for real functions, the method of tangent parabolas. It has order 3. (This method and the previous one are probably the best known order 3 methods for solving nonlinear equations.)

- Convex acceleration of Newton's method or the super-Halley method [7]:

$$z_{n+1} = z_n - \frac{f(z_n)}{2f'(z_n)} \frac{2 - L_f(z_n)}{1 - L_f(z_n)} = z_n - \frac{f(z_n)}{f'(z_n)} \left(1 + \frac{\frac{1}{2} L_f(z_n)}{1 - L_f(z_n)} \right).$$

  This is an order 3 method. (Note that, in [3, p. 248], it is called Halley-Werner's method.)

One group of procedures for solving nonlinear equations are the fixed point methods, which are devoted to solving $F(z) = z$. The best known of these methods is the one that iterates $z_{n+1} = F(z_n)$; it is an order 1 method, and needs a strong hypothesis on F to converge, namely, it requires F to be a contractive function.

An order 2 method for solving an equation $F(z) = z$ is Stirling's fixed point method [3, p. 251 and p. 260]. It starts at a suitable point z_0 and iterates

$$z_{n+1} = z_n - \frac{z_n - F(z_n)}{1 - F'(F(z_n))}.$$

If we want to solve an equation $f(z) = 0$, we can transform it into a fixed point equation. To do this, we can take $F(z) = z - f(z)$. It is then clear that $F(z) = z \Leftrightarrow f(z) = 0$, so we can try to use a fixed point method for F. But this is not the only way: for instance, we can take $F(z) = z - \lambda f(z)$ with $\lambda \neq 0$ a constant (in this way, the already mentioned Whittaker's method appears), or $F(z) = z - \varphi(z) f(z)$ with φ a nonvanishing function. Also, we can isolate z in the expression $f(z) = 0$ in different ways (for instance, if we have $z^3 - z + \tan(z) = 0$, we can isolate $z^3 + \tan(z) = z$ or $\arctan(z - z^3) = z$). Of course, this gives many different fixed point equations $F(z) = z$ for the the same original equation $f(z) = 0$.

Furthermore, when we try to solve $f(z) = 0$ by means of an iterative method $z_{n+1} = \phi(z_n)$, like the ones shown above, and $\{z_n\}_{n=0}^{\infty}$ converges to ζ, it is clear that ζ is a fixed point for ϕ (requiring that ϕ be a continuous function and taking limits in $z_{n+1} = \phi(z_n)$) so, without noticing, we are dealing with fixed point methods.

But it is interesting to check what happens if we merely use the simplest way of taking $F(z) = z - f(z)$ without worrying about any hypothesis. In this way, we have

- (Shifted) Stirling's method:

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n - f(z_n))}.$$

  Its order of convergence is 2.

In all the methods that we have seen until now, the function f or its derivatives are evaluated, in each step of the method, for a single point. There are other techniques for solving nonlinear equations that require the evaluation of f or its derivatives at more than one point in each step. These iterative methods are know as multipoint methods. They are usually employed to increase the order of convergence without computing more derivatives of the function involved. A general study of multipoint methods can be found in [18, Ch. 8 and 9]. Let us look at some multipoint methods.

- Steffensen's method (see [15, p. 198] or [18, p. 178]):

$$z_{n+1} = z_n - \frac{f(z_n)}{g(z_n)}$$

  with $g(z) = \frac{f(z+f(z)) - f(z)}{f(z)}$. This is one of the most simple multipoint methods. The iterative function is generated by a derivative estimation: in Newton's method, for small enough $h = f(z)$, we estimate $f'(z) \approx \frac{f(z+h) - f(z)}{h} = g(z)$. This avoids computing the derivative of f. This is an order 2 method (observe that it preserves the order of convergence of Newton's method).

- Midpoint method (see [18, p. 164] or [3, p. 197]):

$$z_{n+1} = z_n - \frac{f(z_n)}{f'\left(z_n - \frac{f(z_n)}{2f'(z_n)}\right)}.$$

This is an order 3 method.

- Traub-Ostrowski's method (see [18, p. 184] or [3, p. 230]):

$$z_{n+1} = z_n - u(z_n)\frac{f(z_n - u(z_n)) - f(z_n)}{2f(z_n - u(z_n)) - f(z_n)}$$

with $u(z) = \frac{f(z)}{f'(z)}$. Its order of convergence is 4, the highest for the methods that we are studying.

- Jarratt's method [12, 2] (for different expressions, see also [3, p. 230 and p. 234]):

$$z_{n+1} = z_n - \tfrac{1}{2}u(z_n) + \frac{f(z_n)}{f'(z_n) - 3f'(z_n - \frac{2}{3}u(z_n))}$$

where, again, $u(z) = \frac{f(z)}{f'(z)}$. This is also an order 4 method.

- Inverse-free Jarratt's method (see [6] or [3, p. 234]):

$$z_{n+1} = z_n - u(z_n) + \frac{3}{4}u(z_n)h(z_n)\left(1 - \frac{3}{2}h(z_n)\right),$$

with $u(z) = \frac{f(z)}{f'(z)}$ and $h(z) = \frac{f'\left(z - \frac{2}{3}u(z)\right) - f'(z)}{f'(z)}$. Also an order 4 method.

Fractal pictures and comparative tables

We are going to apply the iterative methods that we have seen in the previous section to obtain the complex roots of the functions

$$f(z) = z^3 - 1 \quad \text{and} \quad f^*(z) = \exp(\tfrac{\sin(z)}{100})(z^3 - 1).$$

It is clear that the roots of f^* are the same as the roots of f, that is, 1, $e^{2\pi i/3}$ and $e^{4\pi i/3}$. But the function f^* is much more complicated (i.e., consuming more computer time) to evaluate. Moreover, the successive derivatives of f are easier and easier, and this is just the contrary as a general case. This does not happen with f^*. So, f^* can be a better test to check the speed of these numerical methods in a more general way. (Also, note that many of these iterative methods are also adapted to solve systems of equations or equations in Banach spaces. Here, to evaluate Fréchet derivatives is, usually, very difficult.)

We take a rectangle $D \subset \mathbb{C}$ and we apply the iterative methods starting in "every" $z_0 \in D$. In practice, we take a grid of 1024×1024 points in D and we use these points as z_0. Also, we use two different regions: the rectangle $R_b = [-2.5, 2.5] \times [-2.5, 2.5]$ and a small rectangle near the root $e^{2\pi i/3}$ ($\approx -0.5 + 0.866025i$), the rectangle $R_s = [-0.6, -0.4] \times [0.75, 0.95]$. The first rectangle contains the three roots; the numerical methods starting from a point in R_b can converge to some of the roots or, eventually, diverge. However, R_s is near a root, so it is expected that any numerical method starting there will always converge to the root.

In all these cases, we use a tolerance $\varepsilon = 10^{-8}$ and a maximum of 40 iterations. We denote the three roots as $\zeta_k = e^{2k\pi i/3}$, $k = 0, 1, 2$, and ϕ the iterative method to be used. Then, we take z_0 in the corresponding rectangle and we iterate $z_{n+1} = \phi(z_n)$ up to $|z_n - \zeta_k| < \varepsilon$ for $k = 0, 1$ or 2. If we have not obtained the desired

Table 1. Function f and rectangle R_b.

	Ord	Eff	NC	I/P	T	P/S	I/S
Nw	2	1.41	0.00267	7.52	1	1	1
NwM	2	1.26	0.00381	7.93	1.17	0.857	0.904
CaWh	2	1.41	24.5	18.9	3.23	0.309	0.778
DcaWh	3	1.44	0.125	6.5	1.41	0.711	0.615
Ha	3	1.44	0	4.38	0.901	1.11	0.646
Ch	3	1.44	0.0492	6.27	1.11	0.902	0.752
CaN/sH	3	1.44	0	3.82	0.815	1.23	0.623
Stir	2	1.41	86.6	36.4	4.71	0.212	1.03
Steff	2	1.41	85	35.7	5.79	0.173	0.820
Mid	3	1.44	4.62	6.32	1.1	0.911	0.766
Tr-Os	4	1.59	0	3.69	0.696	1.44	0.705
Ja	4	1.59	0	3.69	0.699	1.43	0.702
IfJa	4	1.59	1.62	7.45	1.41	0.711	0.705

Table 2. Function f and rectangle R_s.

	Ord	Eff	NC	I/P	T	P/S	I/S
Nw	2	1.41	0	2.97	1	1	1
NwM	2	1.26	0	2.97	1.1	0.910	0.910
CaWh	2	1.41	0	3.23	1.39	0.719	0.781
DcaWh	3	1.44	0	2	1.1	0.911	0.613
Ha	3	1.44	0	2	1.03	0.974	0.656
Ch	3	1.44	0	2	0.914	1.09	0.737
CaN/sH	3	1.44	0	2	1.06	0.946	0.636
Stir	2	1.41	0	4.15	1.36	0.733	1.02
Steff	2	1.41	0	3.44	1.42	0.706	0.82
Mid	3	1.44	0	2	0.898	1.11	0.749
Tr-Os	4	1.59	0	1.96	0.925	1.08	0.714
Ja	4	1.59	0	1.96	0.928	1.08	0.712
IfJa	4	1.59	0	1.99	0.969	1.03	0.690

tolerance with 40 iterations, we do not continue and we decide that the iterative method starting at z_0 does not converge to any root.

With these results, combining f and f^* with R_b and R_s we built four tables. In them, the methods are identified as follows: Nw (Newton), NwM (Newton for multiple roots), CaWh (convex acceleration of Whittaker), DcaWh (double convex acceleration of Whittaker), Ha (Halley), Ch (Chebyshev), CaN/sH (convex acceleration of Newton or super-Halley), Stir (Stirling), Steff (Steffensen), Mid (midpoint), Tr-Os (Traub-Ostrowski), Ja (Jarratt), IfJa (inverse-free Jarratt).

For each of them, we show the following information:

- Ord: Order of convergence.
- Eff: Efficiency index.
- NC: No convergent points, as a percentage of the total number of starting points evaluated (which is 1024^2 for every method).
- I/P: Mean of iterations, measured in iterations/point.
- T: Used time in seconds relative to Newton's method (Newton = 1).
- P/S: Speed in points/second relative to Newton's method (Newton = 1).
- I/S: Speed in iterations/second relative to Newton's method (Newton = 1).

TABLE 3. Function f^* and rectangle R_b.

	Ord	Eff	NC	I/P	T	P/S	I/S
Nw	2	1.41	3.06	8.17	1	1	1
NwM	2	1.26	2.86	8.2	1.47	0.681	0.683
CaWh	2	1.41	33.2	19.9	3.58	0.279	0.679
DcaWh	3	1.44	18.1	11	1.88	0.532	0.714
Ha	3	1.44	0.321	4.48	0.918	1.09	0.597
Ch	3	1.44	11.5	9.11	1.56	0.641	0.714
CaN/sH	3	1.44	1.92	4.59	0.907	1.10	0.619
Stir	2	1.41	87.7	36.5	4.04	0.248	1.10
Steff	2	1.41	84.5	35.6	3.39	0.295	1.28
Mid	3	1.44	5.61	6.57	1.21	0.824	0.662
Tr-Os	4	1.59	1.10	4.03	0.677	1.48	0.729
Ja	4	1.59	0.965	3.99	0.777	1.29	0.628
IfJa	4	1.59	19	11.2	1.71	0.584	0.797

TABLE 4. Function f^* and rectangle R_s.

	Ord	Eff	NC	I/P	T	P/S	I/S
Nw	2	1.41	0	2.97	1	1	1
NwM	2	1.26	0	2.97	1.50	0.666	0.666
CaWh	2	1.41	0	3.22	1.67	0.599	0.649
DcaWh	3	1.44	0	2	1.13	0.883	0.594
Ha	3	1.44	0	2	1.10	0.906	0.61
Ch	3	1.44	0	2	1.06	0.944	0.636
CaN/sH	3	1.44	0	2	1.12	0.895	0.602
Stir	2	1.41	0	4.13	1.38	0.724	1.01
Steff	2	1.41	0	3.43	1.06	0.945	1.09
Mid	3	1.44	0	2	1.02	0.979	0.659
Tr-Os	4	1.59	0	1.96	0.909	1.1	0.727
Ja	4	1.59	0	1.96	1.04	0.959	0.634
IfJa	4	1.59	0	1.99	1.05	0.955	0.639

To construct the tables, I used a C++ program in a Power Macintosh 8200/120 computer. In the tables, I show the time and speed relative to Newton's method, so that this will be approximately the same in any other computer. In our computer, the absolute values for Newton's method are the following:

- For Table 1, 137.467 sec, 7627.86 pt/sec and 57336.9 it/sec.
- For Table 2, 59.1667 sec, 17722.4 pt/sec and 52610.2 it/sec.
- For Table 3, 410.683 sec, 2553.25 pt/sec and 20870.6 it/sec.
- For Table 4, 150.083 sec, 6986.63 pt/sec and 20737 it/sec.

In any case, a computer programming language that permits dealing with operations with complex numbers in the same way as for real numbers (such as C++ or Fortran) is highly recommended.

With respect to our time measurements, it is important to note that, for any iterative method $z_{n+1} = \phi(z_n)$, I have written general procedures applicable to generic f and its derivatives. That means, for instance, that when I use f^*, I do not simplify any factor in $\frac{f^*(z)}{(f^*)'(z)}$. Also, if a subexpression of $(f^*)'$ has already been computed in f^* (say, $\sin(z)$) in the generic procedure to evaluate f, its value is not used, but computed again, in the procedure that calculates generic f'. If we were only interested in a particular function f (or if we want a figure in the fastest way), it would be possible to modify the procedure that iterates $z_{n+1} = \phi(z_n)$ for f, adapting and simplifying its expression.

Now, let us go back to the other target of this paper: to visually compare the fractal pictures that appear when we apply different iterative methods for solving an unique equation $f(z) = 0$, where f is a complex function.

In Figures 1 to 12, I show the pictures that appear when we apply the iterative methods to find the roots of the function $f(z) = z^3 - 1$ in the rectangle R_b. I have used strategy (iii) described in the introduction. Respectively, I assign cyan, magenta and yellow for the attraction basins of the three roots 1, $e^{2\pi i/3}$ and $e^{4\pi i/3}$, lighter or darker according to the number of iterations needed to reach the root with the fixed precision required. I mark with black the points $z_0 \in R_b$ for which the corresponding iterative method starting in z_0 does not reach any root with tolerance 10^{-3} in a maximum of 25 iterations.

In the last section of this paper, I show the programs that I have used and similar ones that allow us to generate both grey scaled or color figures. Of course, it is also possible to use the function f^* or the small rectangle R_s (or any other function or rectangle); it will only require small modifications to the programs. This is left for the interested reader.

Although using an ordinary programming language is typically hundreds of times faster, to generate the pictures, it is easier if we employ a computer package with graphics facilities, such us Mathematica, Maple or Matlab. The graphics that I show here were generated with Mathematica 3.0 (see [20]); in the next section, I show the programs used to obtain the Figures.

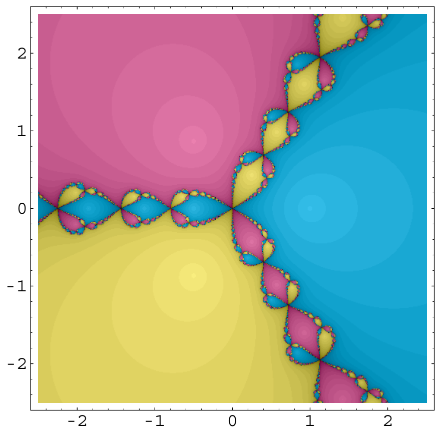

FIGURE 1. Newton's method.

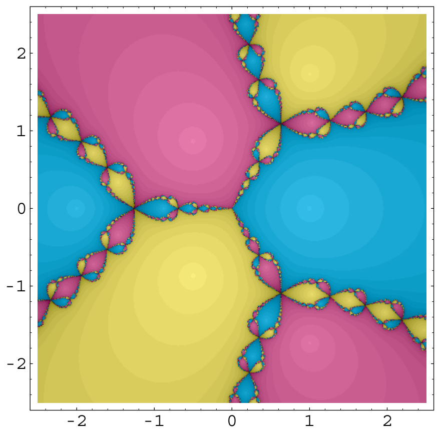

FIGURE 2. Newton's method for multiple roots.

Note that both Traub-Ostrowski's method and Jarratt's method for $f(z) = z^3 - 1$ generate the iterative function $\phi(z) = \frac{1+12z^3+54z^6+14z^9}{6z^2+42z^5+33z^8}$. Then, the fractal figure for both of them is the same (Figure 11), and the same happens for the data of Tables 1 and 2.

In the tables and in the figures, we only have empirical data. From them, and according to the indications given in this paper, we can easily judge the behavior and suitability of any method depending on the circumstances, and which one to choose if we need to solve an equation. This is good entertainment.

Finally, note that Stirling's and Steffensen's methods generate very different results in two ways. First, they are the most demanding with respect to the initial point (in the tables, see the percentage of nonconvergent points and, in the figures,

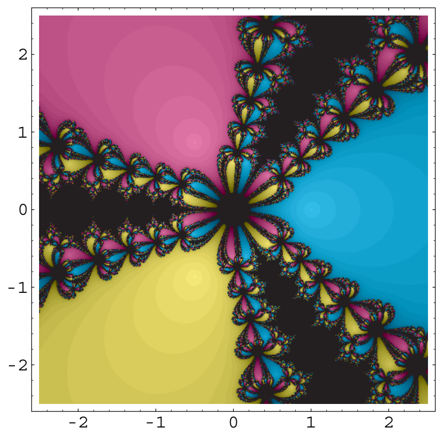

FIGURE 3. Convex acceleration of Whittaker's method.

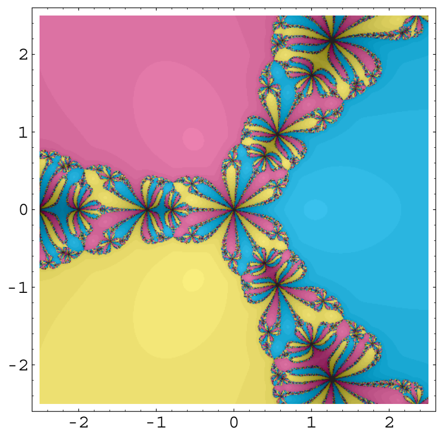

FIGURE 4. Double convex acceleration of Whittaker's method.

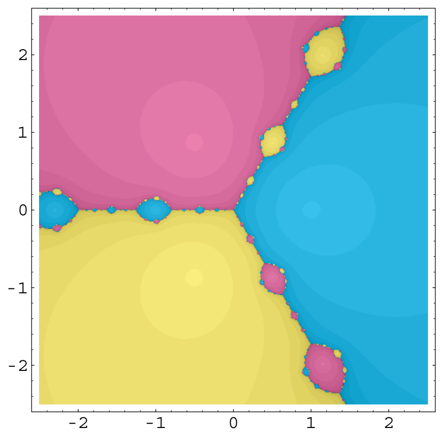

FIGURE 5. Halley's method.

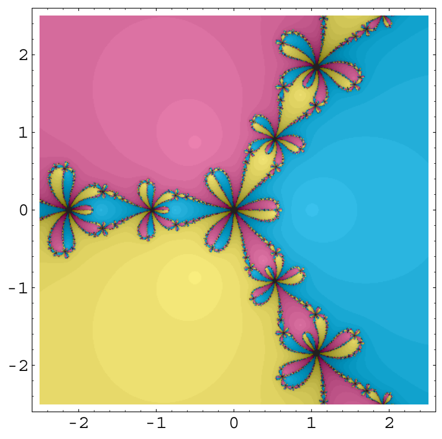

FIGURE 6. Chebyshev's method.

see the black color). And, second, in their graphics, the symmetry of angle $2\pi/3$ that we can observe in the figures corresponding to the other methods does not appear (with respect to the symmetry on fractals, see [1]).

MATHEMATICA PROGRAMS TO GET THE GRAPHICS

In this section, I am going to explain how the Figures of this paper were generated. To do this, I show the Mathematica [20] programs that I have used.

First, we need to define function f and its derivatives. This can be done by using `f[z_] := z^3-1`, `df[z_] := 3*z^2` and `d2f[z_] := 6*z`, but it is faster if we use the compiled versions

```
f = Compile[{{z,_Complex}}, z^3-1];
df = Compile[{{z,_Complex}}, 3*z^2];
d2f = Compile[{{z,_Complex}}, 6*z];
```

Of course, any other function, such us $f^*(z) = \exp(\frac{\sin(z)}{100})(z^3 - 1)$, can be used. If the roots are the same (as happens with f and f^*), we only need to change the previous definitions for `f`, `df` and `d2f`. The figures that appear are somewhat different.

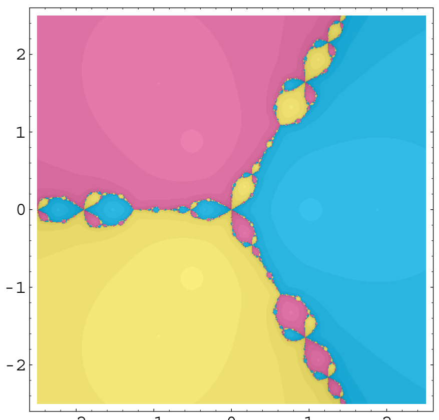

FIGURE 7. Convex acceleration of Newton's method (or super-Halley's method).

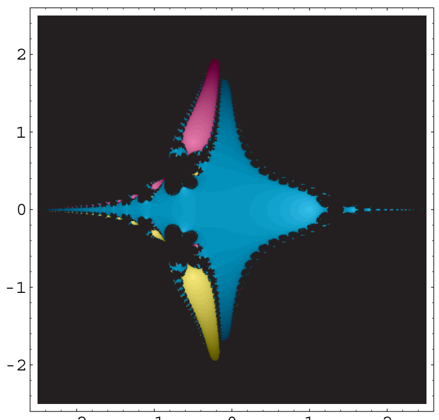

FIGURE 8. (Shifted) Stirling's method.

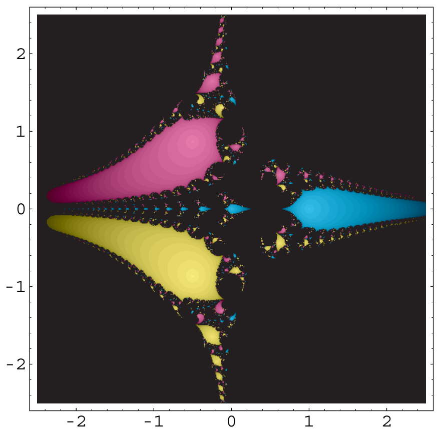

FIGURE 9. Steffensen's method.

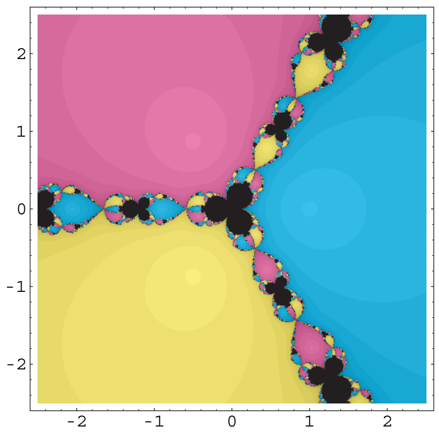

FIGURE 10. Midpoint method.

The three complex roots for f are

```
Do[rootf[k] = N[Exp[2*(k-1)*Pi*I/3]], {k,1,3}]
```

I use the following procedure that identifies which root has been approximated with a tolerance of 10^{-3}, if any,

```
rootPosition = Compile[{{z,_Complex}},
    Which[Abs[z - rootf[1]] < 10.0^(-3), 3,
        Abs[z - rootf[2]] < 10.0^(-3), 2,
        Abs[z - rootf[3]] < 10.0^(-3), 1,
        True, 0],
    {{rootf[_],_Complex}}
    ]
```

We must define the iterative methods, that is, the different $z_{n+1} = \phi(z_n)$. For Newton's method, this would be

```
iterNewton = Compile[{{z,_Complex}}, z-f[z]/df[z]]
```

and, for Halley's method,

```
iterHalley = Compile[{{z,_Complex}},
```

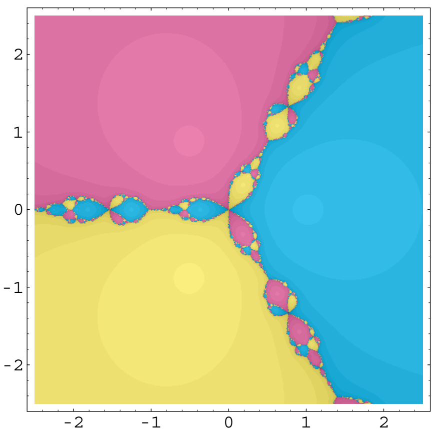

FIGURE 11. Traub-Ostrowski's method and Jarratt's method.

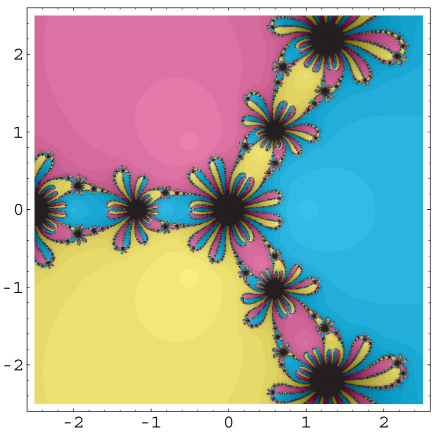

FIGURE 12. Inverse-free Jarratt's method.

```
        Block[{v = df[z]}, z - 1.0 / (v/f[z] - (d2f[z])/(2.0*v))]
        ]
```

(observe that an extra variable `v` is used to evaluate `df[z]` once only). The procedure is similar for all the other methods in this paper.

The algorithm that iterates the function `iterMethod` to see if a root is reached in a maximum of `lim` iterations is the following:

```
    iterAlgorithm[iterMethod_,x_,y_,lim_] :=
        Block[{z,ct,r}, z = x + y I; ct = 0; r = rootPosition[z];
            While[(r == 0) && (ct < lim),
                ++ct; z = iterMethod[z]; r = rootPosition[z]
            ];
            If[Head[r] == Which, r = 0]; (* "Which" unevaluated *)
            Return[r]
        ]
```

Here, I have take into account that, sometimes, Mathematica is not able to do a numerical evaluation of `z`. Then, it cannot assign a value for `r` in `rootPosition`. Instead, it returns an unevaluated `Which`. Of course, this corresponds to nonconvergent points.

We are going to use a limit of 25 iterations and the complex rectangle $[-2.5, 2.5] \times [-2.5, 2.5]$. To do this, I define the following variables:

```
    limIterations = 25;
    xxMin = -2.5;  xxMax = 2.5;  yyMin = -2.5;  yyMax = 2.5;
```

Finally, I define the procedure to paint the figures according the strategy (i) described in the introduction. White, 33 % grey and 66 % grey are used to identify the attraction basins of the three roots 1, $e^{2\pi i/3}$ and $e^{4\pi i/3}$. The points for which the iterative method does not reach any root (with the desired tolerance in the maximum of iterations) are pictured as black. The variable `points` means that, to generate the picture, a `points` $\times$ `points` grid must be used.

```
    plotFractal[iterMethod_,points_] :=
        DensityPlot[iterAlgorithm[iterMethod,x,y,limIterations],
            {x, xxMin, xxMax}, {y, yyMin, yyMax},
            PlotRange->{0,3}, PlotPoints->points, Mesh->False
        ] // Timing
```

Note that `// Timing` at the end allows us to observe the time that Mathematica employs when `plotFractal` is used.

Then, a graphic is obtained in this way (the example is a black and white version of Figure 1):

```
plotFractal[iterNewton, 256]
```

When we use the functions that we have defined, overflow and underflow errors can happen (for instance, in Newton's method, $f'(z)$ can be null and then we are dividing by zero, although it is not the only problem). Mathematica informs us of such circumstances; to avoid it, use the following before calling `plotFractal`:

```
Off[General::ovfl];    Off[General::unfl];
Off[Infinity::indet]
```

Also, the previous problems, and some other ones, sometimes force Mathematica to use a noncompiled version of the functions. Again, Mathematica informs us of that circumstance; to avoid it, use

```
Off[CompiledFunction::cccx];    Off[CompiledFunction::cfn];
Off[CompiledFunction::cfcx];    Off[CompiledFunction::cfex];
Off[CompiledFunction::crcx];    Off[CompiledFunction::ilsm]
```

Perhaps some other `Off` are usefull according the function `f` and the complex rectangle used.

To obtain color figures, I use a slightly different procedure to identify which root has been approximated; this is so because we also want to know how many iterations are necessary to reach the root. We use the following trick: in the output, the integer part corresponds to the root and the fractional part is related to the number of iterations.

```
iterColorAlgorithm[iterMethod_,x_,y_,lim_] :=
    Block[{z,ct,r}, z = x + y I; ct = 0; r = rootPosition[z];
        While[(r == 0) && (ct < lim),
            ++ct; z = iterMethod[z]; r = rootPosition[z]
        ];
        If[Head[r] == Which, r = 0]; (* "Which" unevaluated *)
        Return[N[r+ct/(lim+0.001)]]
    ]
```

To assign the intensity of the color of a point, I take into account the number of iterations used to reach the root when the iterative method starts at that point. I use cyan, magenta and yellow for the points that reach, respectively, the roots 1, $e^{2\pi i/3}$ and $e^{4\pi i/3}$; and black for nonconvergent points. To do this, I use

```
colorLevel = Compile[{{p,_Real}}, 0.4*FractionalPart[4*p]]
```

and

```
fractalColor[p_] :=
    Block[{pp = colorLevel[p]},
        Switch[IntegerPart[4*p],
            3, CMYKColor[0.6+pp,0.,0.,2*pp],
            2, CMYKColor[0.,0.6+pp,0.,2*pp],
            1, CMYKColor[0.,0.,0.6+pp,2*pp],
            0, CMYKColor[0.,0.,0.,1.]
        ]
    ]
```

(In the internal behavior of Mathematica, when a function is going to be pictured with `DensityPlot`, it is scaled to $[0,1]$. However, `iterColorAlgorithm` has a range of $[0,4]$; this is the reason for using `4*p` in some places in `colorLevel` and fractalColor. Also, note that colorLevel can be changed to modify the intensity of the colors; for other graphics, it is a good idea to experiment by changing the parameters to get nice pictures.)

Finally, a color fractal will be pictured by calling the procedure

```
plotColorFractal[iterMethod_,points_] :=
    DensityPlot[
        iterColorAlgorithm[iterMethod,x,y,limIterations],
        {x, xxMin, xxMax}, {y, yyMin, yyMax},
        PlotRange->{0,4}, PlotPoints->points, Mesh->False,
        ColorFunction->fractalColor
    ] // Timing
```

For instance,

```
plotColorFractal[iterNewton, 256]
```

is just Figure 1.

Families of iterative methods

There are many iterative methods for solving nonlinear equations in which there a parameter appears, which are usually called families of iterative methods.

One of the best known is the Chebyshev-Halley family

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)} \left(1 + \frac{1}{2} \frac{L_f(z_n)}{1 - \beta L_f(z_n)} \right),$$

with β a real parameter. These are order 3 methods for solving the equation $f(z) = 0$. Particular cases are $\beta = 0$ (Chebyshev's method), $\beta = 1/2$ (Halley's method), and $\beta = 1$ (super-Halley's method). When $\beta \to -\infty$, we get Newton's method. This family was studied by Werner in 1980 (see [19]), and can also be found in [3, p. 219] and [10]. It is interesting to note that any iterative process given by the expression

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)} H(L_f(z_n)),$$

where function H satisfies $H(0) = 0$, $H'(0) = 1/2$ and $|H''(0)| < \infty$, generates an order 3 iterative method (see [8]). In this way, the Chebyshev-Halley family appears by taking $H(x) = 1 + \frac{1}{2}\frac{x}{1-\beta x}$.

A multipoint family (see [18, p. 178]) is

$$z_{n+1} = z_n - \frac{f(z_n)}{g(z_n)}$$

with $g(z) = \frac{f(z+\beta f(z)) - f(z)}{\beta f(z)}$ and β an arbitrary constant ($\beta = 1$ is Steffensen's method). Its order of convergence is 2.

An order 4 multipoint family was studied by King [14] (see also [3, p. 230]):

$$z_{n+1} = z_n - u(z_n) - \frac{f(z_n - u(z_n))}{f'(z_n)} \frac{f(z_n) + \beta f(z_n - u(z_n))}{f(z_n) + (\beta - 2) f(z_n - u(z_n))},$$

where β is an arbitrary real number and $u(z) = \frac{f(z)}{f'(z)}$. It generalizes Traub-Ostrowski's method (which is the particular case $\beta = 0$).

Finally, let us cite another order 4 multipoint family

$$z_{n+1} = z_n - u(z_n) + \frac{3}{4} u(z_n) h(z_n) \frac{1 + \beta h(z_n)}{1 + (\frac{3}{2} + \beta) h(z_n)},$$

where β is a parameter and u, h denote $u(z) = \frac{f(z)}{f'(z)}$ and $h(z) = \frac{f'\left(z - \frac{2}{3}u(z)\right) - f'(z)}{f'(z)}$. Here, for $\beta = 0$, we get Jarratt's method (actually, in [12] a different family appears;

the method that we are calling Jarratt's method is really a particular case for both families). Moreover, for $\beta = -3/2$, we get the so-called inverse-free Jarratt's method.

Uniparametric iterative methods offer an interesting graphic possibility: to show pictures in movement. We take a fixed function and a fixed rectangle, and we represent the fractal pictures for many values of the parameter. This then generates a nice moving image that shows the evolution of the fractal images when the parameter varies. Unfortunately, it is not possible to show moving images in a paper. To generate them in a computer, we can use small modifications of the Mathematica programs from the previous section, using also the Mathematica commands `Animate` or `ShowAnimation`. Later, it is possible to export these images in QuickTime format (so that Mathematica will not be necessary to see them). Of course, this requires a large quantity of computer time, but computers are becoming faster and faster, so this is not a very big problem.

References

1. C. Alexander, I. Giblin and D. Newton, Symmetry groups on fractals, *The Mathematical Intelligencer* **14** (1992), no. 2, 32–38.
2. I. K. Argyros, D. Chen and Q. Qian, The Jarratt method in Banach space setting, *J. Comput. Appl. Math.* **51** (1994), 103–106.
3. I. K. Argyros and F. Szidarovszky, *The theory and applications of iteration methods*, CRC Press, Boca Raton, FL, 1993.
4. W. Bergweiler, Iteration of meromorphic functions, *Bull. Amer. Math. Soc. (N. S.)* **29** (1993), 151–188.
5. P. Blanchard, Complex analytic dynamics on the Riemann sphere, *Bull. Amer. Math. Soc. (N. S.)* **11** (1984), 85–141.
6. J. A. Ezquerro, J. M. Gutiérrez, M. A. Hernández and M. A. Salanova, The application of an inverse-free Jarratt-type approximation to nonlinear integral equations of Hammerstein-type, *Comput. Math. Appl.* **36** (1998), 9–20.
7. J. A. Ezquerro and M. A. Hernández, On a convex acceleration of Newton's method, *J. Optim. Theory Appl.* **100** (1999), 311–326.
8. W. Gander, On Halley's iteration method, *Amer. Math. Monthly* **92** (1985), 131–134.
9. W. Gautschi, *Numerical analysis: An introduction*, Birkhäuser, Boston, 1997.
10. J. M. Gutiérrez and M. A. Hernández, A family of Chebyshev-Halley type methods in Banach spaces, *Bull. Austral. Math. Soc.* **55** (1997), 113–130.
11. M. A. Hernández, An acceleration procedure of the Whittaker method by means of convexity, *Zb. Rad. Prirod.-Mat. Fak. Ser. Mat.* **20** (1990), 27–38.
12. P. Jarratt, Some fourth order multipoint iterative methods for solving equations, *Math. Comp.* **20** (1966), 434–437.
13. D. Kincaid and W. Cheney, *Numerical analysis: Mathematics of scientific computing*, 2nd ed., Brooks/Cole, Pacific Grove, CA, 1996.
14. R. F. King, A family of fourth order methods for nonlinear equations, *SIAM J. Numer. Anal.* **10** (1973), 876–879.
15. J. M. Ortega and W. C. Rheinboldt, *Iterative solution of nonlinear equations in several variables*, Monographs Textbooks Comput. Sci. Appl. Math., Academic Press, New York, 1970.
16. H. O. Peitgen and P. H. Richter, *The beauty of fractals*, Springer-Verlag, New York, 1986.
17. M. Shub, Mysteries of mathematics and computation, *The Mathematical Intelligencer* **16** (1994), no. 1, 10–15.
18. J. F. Traub, *Iterative Methods for the Solution of Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1964.
19. W. Werner, Some improvements of classical iterative methods for the solution of nonlinear equations, in *Numerical Solution of Nonlinear Equations* (Proc., Bremen, 1980), E. L. Allgower, K. Glashoff and H. O. Peitgen, eds., *Lecture Notes in Math.* **878** (1981), 427–440.
20. S. Wolfram, *The Mathematica Book*, 3rd ed., Wolfram Media/Cambridge University Press, 1996.
21. J. W. Neuberger, Continuous Newton's method for polynomials, *The Mathematical Intelligencer* **21** (1999), no. 3, 18–23.

Departamento de Matemáticas y Computación, Universidad de La Rioja, Edificio J. L. Vives, Calle Luis de Ulloa s/n, 26004 Logroño, Spain

E-mail address: jvarona@dmc.unirioja.es

URL: http://www.unirioja.es/dptos/dmc/jvarona/welcome.html